SocketTools10

.NET Edition

Developer's Guide and Technical Reference

Version 10.0.1468.2518

Introduction

The SocketTools .NET Edition is a collection of managed code classes that simplify the task of developing TCP/IP networking applications in Visual Studio .NET using any of the available programming languages such as Visual Basic and C#. The .NET Edition is ideal for the developer who requires the flexibility, ease of use and rapid development features of a component without the complexities of working with the native socket class or in-depth knowledge of how the various Internet protocols are implemented. The SocketTools .NET Edition consists of fourteen core networking classes which can be used to develop applications that meet a wide range of needs. SocketTools covers it all, including uploading and downloading files, sending and retrieving email, remote command execution, terminal emulation, and much more.

The SocketTools .NET Edition includes support for the industry standard Transport Security Layer (TLS) and Secure Shell (SSH) protocols which are used to ensure that data exchanged between the local system and a server is secure and encrypted. The .NET Edition implements the major secure protocols such as HTTPS, FTPS, SFTP, SMTPS, POP3S, IMAPS and more. Your data is protected with TLS 1.2 using 256-bit encryption and full support for client certificates. SocketTools also includes an FTP and HTTP server component, as well as a general purpose TCP server component that can be used to create custom server applications. There's no need for you to understand the details of certificate management, data encryption or how the security protocols work. All it takes is a few lines of code to enable the security features, and SocketTools handles the rest.

For developers who have used the ActiveX version of SocketTools, you'll immediately find yourself in familiar territory. The SocketTools .NET class interface has properties, methods and events that are very similar to the control that you've used in languages like Visual Basic 6.0. It is important to keep in mind that SocketTools .NET is a managed code class, not a wrapper around the ActiveX control, so there will be some inherent differences. However, whenever possible the class interface was designed to make the transition from the ActiveX control as easy as possible.

The following are just some of the features in the SocketTools 10 .NET Edition:

- Support for Windows 10, Visual Studio 2019 and .NET 5.0
- Managed code class written in C# that can be used with any .NET language
- An interface that is very similar to the components in the SocketTools ActiveX Edition
- Support for both synchronous and asynchronous network connections
- Includes components that can be used to create custom client and server applications
- Provides cloud-based application storage and geographical IP location services
- Support for the standard TLS 1.2 protocol with 256-bit AES encryption
- Support for both implicit and explicit TLS connections
- Support for the SSH protocol and integrated support for SFTP as part of the FTP class
- Support for standard and secure proxy servers using FTP and HTTP
- Support for using client and server certificates in PKCS12 format
- Thread-safe implementation with full support for multithreaded applications
- An extensive Developer's Guide and online Technical Reference
- Easy redistribution for any number of applications and end users

Developer's Guide

To help you get started using SocketTools, the Developer's Guide covers a variety of programming topics related to SocketTools, as well an overview of each of the controls included in the product. Even if you have experience working with previous versions of SocketTools, we recommend that you review the Developer's Guide. If you are using a language other than Visual Basic, you'll also find some very helpful information about how to make the most of SocketTools in other programming languages such as Visual C#.NET and Visual C++.

Technical Reference

The Technical Reference provides extensive documentation on all of the functions in each of the SocketTools controls. It's here that you'll find information on the various properties, methods and events provided by the component. If it is your first time using a particular class, we recommend that you first read the overview of that class in the Developer's Guide.

Licensing Information

The SocketTools .NET Edition License Agreement provides you with a single developer license and the right to redistribute the class libraries (assemblies) included with this product without any additional royalties or runtime licensing fees.

Evaluation Licenses

When you install SocketTools, you are given the option of entering a serial number or proceeding with the installation without a serial number. If you install SocketTools without a serial number, an evaluation development license will be created which is valid for a period of thirty (30) days from the date of installation. The product is fully functional during this evaluation period; however the SocketTools components may not be redistributed to third-parties. After the evaluation period has ended, you must either purchase a development license or remove SocketTools from your computer system.

Runtime Licensing

When you install SocketTools with a serial number, a runtime license key will be automatically generated for you and stored in a file named SocketToolsLicense.cs in the Include folder where you've installed the product. There are similarly named files for other languages, such as SocketToolsLicense.vb. These files define the SocketTools runtime licensing key which must be passed to the Initialize method in the classes that you are using. More information about that utility is provided below.

The runtime license key is a null terminated string that is unique to your licensed copy of SocketTools. The runtime license key is not the same as your serial number and should only be embedded in your compiled application. If you provide source code for your product, you cannot include the runtime key with the source code. The same runtime license key should be used for all of the .NET assemblies.

If you install SocketTools with an evaluation license, then the runtime license key will be defined as an empty string. This will allow the assemblies to function on a system with a valid evaluation license, but they will not function on any other system. You must purchase a license and generate a runtime license key before redistributing an application which uses one or more of the SocketTools assemblies.

License Manager

Included with your copy of SocketTools is a License Manager utility. This program enables you to see what components have been installed and registered on your system, as well as display information about your SocketTools license. If you need to create a new runtime license key, you can use this utility to do so. Select **License** | **Header File** from the menu and choose the type of file that you wish to create. For more information about how the License Manager can be used, please refer to the online help file that is included with the utility.

Product Evaluation

If you install SocketTools without registering a serial number, the product will be installed with an evaluation license that is valid for a period of thirty (30) days. During this trial period, the SocketTools controls are fully functional and can be used on the development system where the product was installed. If you need to extend the evaluation period, please contact the Catalyst Development sales office by email at sales@sockettools.com or by telephone at 1-760-228-9653, Monday through Friday during normal business hours.

Redistribution Restrictions

When using an evaluation copy of SocketTools, you cannot redistribute the controls to another system. If you build an application using an evaluation license, it will function correctly on the development system but will fail with an error on any system that does not have a license. Once you have purchased a development license, you should recompile your application before redistributing it to an end-user. If you need to test your application on another system during the evaluation period, you must install an evaluation copy of SocketTools on that system.

Runtime Licensing

When you purchase a development license, a runtime license key will be generated for you which will be included in your applications. Normally this runtime key is managed automatically when the control is placed on a form or referenced in a project. However, there are situations in which the key must be explicitly passed to the control's Initialize method. In all cases, if the product is installed as an evaluation copy, the runtime license key will be defined as an empty string. If you have previously installed an evaluation copy of SocketTools and then purchased a license, you can create the runtime license key using the License Manager utility.

SocketTools Upgrade Information

This section will help you upgrade an application written using a previous version of the SocketTools .NET Edition. In most cases, the modifications required will be minimal and may only require updating your project references and recompiling the program. However, it is recommended that you review this entire guide so that you understand what changes were made and how those changes can be implemented in your software.

Supported Platforms

SocketTools 10 is supported on Windows 7, Windows Server 2008 R2 and later versions. Earlier versions of the operating system, including Windows 2000, Windows XP and Windows Vista are no longer supported by Microsoft and are not recommended for use with SocketTools.

Developers who are redistributing applications which target Windows 10 or Windows Server 2019 should upgrade to ensure compatibility with the platform and current development tools. Secure connections on Windows XP and Windows Vista has been deprecated because those platforms do not support TLS 1.2 and most services will no longer accept connections from a client using SSL 3.0 or TLS 1.0.

Development Tools

The SocketTools .NET components may be used with Visual Studio 2010 and later versions. If you are developing using Visual Studio 2010, use the .NET 4.0 assemblies. For Visual Studio 2012 and later versions, you can use the .NET 4.5 assemblies. If you are targeting a later version of the Framework, such as .NET 4.7.2 or .NET 4.8, you should reference the .NET 4.5 assemblies, which are compatible with those versions. The .NET 5.0 Framework is supported with Visual Studio 2019.

If you are developing on Windows 7 or Windows 8.1, it is recommended that you use Visual Studio 2010 or a later version. Earlier versions of Visual Studio may require that you use those development tools with elevated privileges.

If you are developing on Windows 10, it is recommended you use Visual Studio 2017 or later versions. SocketTools 10 includes assemblies which target the .NET 4.5 Framework as well as .NET 5.0.

Upgrading From SocketTools 9.5

If you are upgrading from version 9.5, applications will be source code compatible with the SocketTools 10 .NET classes. In most cases, all you will need to do is install the current version, update your project references and recompile your application. While the assembly modules retain the same name, the platform specific 32-bit and 64-bit interop libraries have changed to **SocketTools10.Interop.dll**. This allows applications created using SocketTools 10 to co-exist with applications created using earlier versions of SocketTools.

Your runtime license key has changed for SocketTools 10, which will require you to define the new key in your application when calling the class **Initialize** method. As with previous versions of SocketTools, you can use the License Manager utility to generate a file which contains the runtime key you should use. The SocketTools 9 and earlier runtime license keys are not valid for the version 10 classes and an error will be returned if an invalid runtime key is specified.

With SocketTools 10, secure connections will use TLS 1.2. By default, the .NET components will not support connections to servers which use older, less secure versions of TLS or any version of SSL. They will also no longer use weaker cipher suites that incorporate insecure algorithms, such as RC4 or MD5. For applications that require secure connections, it is recommended you use the current build of Windows 10 or Windows Server with all security updates applied.

It is possible to force the class to use earlier versions of TLS for backwards compatibility with older servers. This is done by explicitly setting the **SecureProtocol** property to specify the protocol version required.

However, this is not generally recommended because using an older version of TLS (or any version of SSL) may cause servers to immediately reject the connection attempt.

Upgrading From SocketTools 8.0

If you are upgrading from SocketTools 8.0 and earlier versions, most applications will be source code compatible and not require significant changes to existing code. However, it should be noted that the path to the common assembly folder has changed and earlier versions of SocketTools did not provide assemblies which targeted the .NET 4.5 or later framework versions. The SocketTools 10 assemblies are now installed in the folder **C:\Program Files (x86)\Common Files\SocketTools\10.0\Assemblies** for each version of the .NET Framework which is supported.

The runtime license key has changed for SocketTools 10, which will require you to define the new key in your application when calling the class **Initialize** method. As with previous versions of SocketTools, you can use the License Manager utility to generate a file which contains the runtime key you should use. The earlier runtime license keys are not valid for the version 10 classes and an error will be returned if an invalid runtime key is specified.

Interop Libraries

The platform specific interop libraries have been updated for SocketTools 10 and must be installed when deploying your application. For more information, refer to the Redistribution section.

File Name	Description
SocketTools10.Interop.dll	The SocketTools runtime library. This library is shared by all of the SocketTools and SocketTools class libraries. This library should not be referenced directly in the project, but must be included in the application installation package. It is recommended you install this file in the appropriate Windows system folder.
SocketTools10.TraceLog.dll	A debugging library used to generate log files which record the low-level networking functions called and the data exchanged. This library only needs to be distributed if the debugging features of the class are used. This library should not be referenced directly in the project, but should be included in the application installation package. It is recommended that you install this file in the appropriate Windows system folder.

Component Redistribution

The SocketTools license permits the use of the .NET assemblies to build application software and redistribute that software to end-users. There are no restrictions on the number of products in which the classes may be used. However, if it has been installed with an evaluation license, any products built using SocketTools cannot be redistributed to another system until a licensed copy of the toolkit has been purchased.

Supported Platforms

SocketTools .NET is supported on Windows 32-bit and 64-bit desktop and server platforms. The minimum required desktop platform is Windows 7 with Service Pack 1 (SP1) installed. The minimum required server platform is Windows Server 2008 R2 with Service Pack 1 (SP1) installed. It is recommended that the current service pack be installed for the operating system, along with the latest Windows updates available from Microsoft. Some features may require Windows 8.1 or later versions of the platform. When this is the case, it will be noted in the documentation.

Windows 2000, Windows XP and Windows Vista are no longer supported by Microsoft. SocketTools is designed for Windows 7 as the minimum operating system version and may not work correctly on earlier versions of Windows.

SocketTools .NET includes assemblies which target .NET Core 2.1 LTS (Long Term Support) and .NET Core 3.1. You can use these assemblies with .NET Core and .NET Standard projects, however it is important to note that SocketTools is only supported on Windows desktop and server platforms. It is not currently supported for other architectures. If your project is targeting .NET Core 2.1 LTS you will have access to all of the SocketTools assemblies except for the SocketTools.Terminal class which is a graphical control. The required support for graphical controls was added in .NET Core 3.0 and is supported with with the .NET Core 3.1 assemblies.

This version of SocketTools includes support for .NET 5.0, which is effectively a merge of .NET 4.8 and .NET Core 3.1 into a single framework. Microsoft released .NET 5.0 in November 2020 and is included with the latest release of Visual Studio 2019.

Supported Development Tools

SocketTools .NET includes assemblies that can be used with Visual Studio with support for .NET Framework version 2.0 through version 5.0. Other .NET development tools such as Embarcadero RAD Studio and SharpDevelop are also supported. It is recommended that you install all service packs and updates that are available. Development is only supported on Windows platforms and is not currently supported on the Mono implementation of the Common Language Runtime.

Component Files

For those applications created using SocketTools .NET, the appropriate class libraries (assemblies) must be distributed along with the application. The following component files can be included with your application:

Assembly File Name	Description
SocketTools.DnsClient.dll	The Domain Name Services component which provides domain name services for the application. This assembly is added as a reference in the project.
SocketTools.FileEncoder.dll	The File Encoding and Decoding component which provides functions to encode, compress and encrypt files. This assembly is added as a reference in the project.

SocketTools.FileTransfer.dll	The FileTransfer assembly which provides a common interface to the File Transfer Protocol and Hypertext Transfer Protocol. This assembly should be added as a reference in the project.
SocketTools.FtpClient.dll	The File Transfer Protocol client component which enables an application to upload and download files, as well as manage files on the server. This assembly is added as a reference in the project.
SocketTools.FtpServer.dll	The File Transfer Protocol server component which enables an application to accept connections from standard FTP clients which can transfer and manage files on the system. This assembly is added as a reference in the project.
SocketTools.HttpClient.dll	The Hypertext Transfer Protocol client component which enables an application to transfer files and query a web server. This assembly is added as a reference in the project.
SocketTools.HttpServer.dll	The Hypertext Transfer Protocol server component which enables an application to accept connections from HTTP clients, such as web browsers. This assembly is added as a reference in the project.
SocketTools.IcmpClient.dll	The Internet Control Message Protocol component which provides ping and traceroute functionality for an application. This assembly is added as a reference in the project.
SocketTools.ImapClient.dll	The Internet Message Access Protocol component which enables an application to retrieve mail messages from a server and manage one or more mailboxes remotely. This assembly is added as a reference in the project.
SocketTools.InternetDialer.dll	The Remote Access Services component which enables an application to establish a dial-up networking connection using PPP or SLIP. This assembly is added as a reference in the project.
SocketTools.InternetMail.dll	The InternetMail assembly which provides a client interface to the Post Office Protocol, Internet Message Access Protocol and Simple Mail Transfer Protocol. This assembly should be added as a reference in the project.
SocketTools.InternetServer.dll	The Internet Server component which provides the framework for implementing an event-driven multithreaded server application. This assembly is added as a reference in the project.
SocketTools.MailMessage.dll	The Mail Message component which enables an application to create, store and parse email messages. This assembly is added as a reference in the project.
SocketTools.NetworkTime.dll	The Network Time Services component which enables an application to query a time server to determine the current time. This assembly is added as a reference in the project.

SocketTools.NewsFeed.dll	The Syndicated News Feed component which enables an application to load and parse a news feed, either from the local system or a server. This assembly is added as a reference in the project.
SocketTools.NntpClient.dll	The Network News Transfer Protocol component which enables an application to download and process messages from a news server. This assembly is added as a reference in the project.
SocketTools.PopClient.dll	The Post Office Protocol component which enables an application to retrieve messages from a mail server. This assembly is added as a reference in the project.
SocketTools.RshClient.dll	The Remote Command Shell component which enables an application to execute commands on a server, returning the output of the command to the program. This assembly is added as a reference in the project.
SocketTools.SmtpClient.dll	The Simple Mail Transfer Protocol component which enables an application to submit an email message for delivery to a mail server. This assembly is added as a reference in the project.
SocketTools.SocketWrench.dll	The SocketWrench (Windows Sockets) component which provides a general purpose networking interface, allowing applications to connect to other systems and exchange data using the TCP/IP protocol. This assembly is added as a reference in the project.
SocketTools.SshClient.dll	The Secure Shell component which enables an application to establish a secure, encrypted connection with a server providing either an interactive terminal session or the ability to execute commands remotely. This assembly is added as a reference in the project.
SocketTools.TelnetClient.dll	The Telnet Protocol component which enables an application to connect to a server, providing an interactive terminal session. This assembly is added as a reference in the project.
SocketTools.Terminal.dll	The Terminal Emulation component which provides VT-100 and VT-220 terminal emulation services for an application, typically used for interactive sessions with a TELNET or SSH server. This control is added as a visual component to the application.
SocketTools.TextMessage.dll	The TextMessage component which enables an application to send SMS text messages through an SMTP gateway service. This assembly is added as a reference in the project.
SocketTools.WebLocation.dll	The WebLocation component provides information about the physical location of the current computer system based on its external IP address. This assembly is added as a reference in the project.

SocketTools.WebStorage.dll	The WebStorage component provides private cloud storage for uploading and downloading shared data files which are available to your application. This is primarily intended for use by developers to store configuration information and other data generated by the application. This assembly is added as a reference in the project.
SocketTools.WhoisClient.dll	The Whois Protocol component which enables an application to query a server for registration information about a specific domain name. This assembly is added as a reference in the project.
SocketTools10.Interop.dll	The SocketTools runtime library. This library is shared by all of the SocketTools and SocketTools class libraries. This library should not be referenced directly in the project, but must be included in the application installation package. It is recommended you install this file in the appropriate Windows system folder.
SocketTools10.TraceLog.dll	A debugging library used to generate log files which record the low-level networking functions called and the data exchanged. This library only needs to be distributed if the debugging features of the class are used. This library should not be referenced directly in the project, but should be included in the application installation package. It is recommended that you install this file in the appropriate Windows system folder.

Version Information

The SocketTools components have embedded information which provides version information to an installation utility. This information, called the version resource, specifies the assembly's version number among other things. If you are using a third-party or in-house installation program, it is extremely important that the program knows how to use this information. You should never overwrite a newer (higher) version of the SocketTools assembly with an older (lower) version or build number.

Installation Directory

It is recommended that you install the SocketTools assemblies in the same directory as the application executable on the target system. It is not recommended that you install the assemblies in the Global Assembly Cache (GAC) or in the Windows system folder. The assemblies are built to target both the x86 and x64 platforms and can be used on either type of system. The SocketTools10.Interop.dll runtime library is platform specific and it is recommended that you install it in the Windows system folder. This is a shared library that is used by all of the SocketTools assemblies and should not be referenced directly in your project.

If you are deploying your application to a system running the 32-bit version of Windows, you should install the 32-bit version of the SocketTools10.Interop.dll library in the \Windows\System32 folder. If you are deploying your application to a system running the 64-bit version of Windows, you should install the 32-bit version of the runtime library in the \Windows\Syswow64 folder, and the 64-bit version of the library in the \Windows\System32 folder. If you are using the **Trace** related properties in your application, you should also include SocketTools10.TraceLog.dll library in your installation package and install it in the same folder as the runtime library.

The installer should always perform version checking to ensure that it is not overwriting a newer version of the runtime library with an older version. If your installer package creates a 32-bit executable and you're deploying a 64-bit application, the installer must be capable of detecting that it is running on a 64-bit system and can disable filesystem redirection to ensure that the 64-bit libraries are installed in the correct location. Consult the documentation for your installer to determine if it is 64-bit compatible.

Windows Install Packages

To help simplify deployment, SocketTools includes MSI (Windows Installer) packages you can use to install the SocketTools .NET interop libraries on end-user systems. These packages are found in the **Redist** folder where you've installed SocketTools.

Package Name	Description
cstools10_interop_x86.msi	SocketTools 10 .NET redistributable libraries for 32-bit applications. This installer is what developers should use if they are targeting the x86 platform and want their software to run on both 32-bit and 64-bit versions of Windows.
cstools10_interop_x64.msi	SocketTools 10 .NET redistributable libraries for 64-bit applications. This installer should only be used if 64-bit development tools were used to build the application, and can only be installed on 64-bit versions of Windows.

If you're redistributing a 32-bit application, then all you need is the x86 installer package. If you're redistributing a 64-bit application, then you need the x64 installer package. If your application is built to target both platforms (Any CPU), then you will need both x86 and x64 packages. The installer packages will make sure the interop libraries are installed in the correct shared Windows folders and will perform the appropriate version checking.

If you have your own installer for your software, then you can redistribute those MSI packages with your installation and use the **msiexec** command to perform the installation. For example, this would install and register the 32-bit .NET interop libraries with no UI displayed:

```
msiexec /qn /I cstools10_interop_x86.msi
```

For the complete list of command line options for **msiexec**, refer to the Windows Dev Center documentation.

Strong Name Signing

The SocketTools assemblies are given what is known as a "strong name", which means they have been signed with a cryptographic public/private key pair which ensures that the assembly has not been tampered with. However, if you are redistributing an updated version of the assembly, it requires that you also recompile your application and include it in the distribution. Unlike standard Windows dynamic link libraries, you cannot simply redistribute the assembly by itself because a strong-name signed assembly requires that the signature of the assembly, which includes the version number, match exactly what the application was built with. This is a feature in .NET designed to ensure that the application is using the correct version of the assembly.

Technical Support

Catalyst Development is committed to providing quality technical support for our products and we offer several different support options designed to meet the needs of our customers. Technical support by email is available for installation, development and redistribution issues related to the purchased product. There are also paid support options available for customers who require additional assistance.

Standard Support

Registered developers have access to a variety of free technical support resources and we always encourage developers to review our online documentation and knowledge base to determine if the question has already been answered.

Frequently Asked Questions

A collection of answers to the most frequently asked questions about a product. General questions about features, functionality and platform compatibility are answered here. The product FAQ is also recommended reading for any developer who is evaluating our software.

Knowledge Base

A searchable online database of solutions to hundreds of common technical questions and problems. The articles provide detailed information, including background information, workarounds and the availability of updates to resolve the problem. This is the first place that most developers should check to determine if the question or problem that they're having has already been addressed.

Online Documentation

A comprehensive collection of online help, tutorials and whitepapers for our products. Our online help is useful to evaluators who are interested in learning about how our components work and for developers who would like access to the most current reference material.

Release Notes

Information about the latest changes, improvements and corrections made to the current version SocketTools. The release notes can reflect changes that affect all SocketTools editions, as well as updates to a component in a specific edition. If you are upgrading from a previous release, it's recommended that you review the release notes.

Priority Support

For developers who require additional support, Priority Support offers a guaranteed, priority response to technical support issues on the same business day. Corrections which require a source code change and/or documentation change to resolve a problem will be made available as a hotfix at no additional charge, and whenever there is a new product update or hotfix, you will be automatically notified by email.

Premium Support

For developers who have critical support needs, an annual Premium Support agreement offers both telephone and email support, and a guaranteed four hour response time during business hours. This support option also includes all of the benefits of priority support, including hotfixes, source code analysis and assistance with example code. In addition, Premium Support also includes free upgrades if a new version of the product is released while your support agreement is active, ensuring that you're always working with the latest version.

License Agreement

This License Agreement is a legal agreement between you, either as an individual or a single entity ("Developer"), and Catalyst Development Corporation ("Catalyst") for the software product identified as "SocketTools .NET Edition" ("Software" or "Software Product"). The Software Product includes executable programs, redistributable modules, controls, and dynamic link libraries ("Components" or "Software Components"), electronic documentation, and may include associated media and printed materials.

Installing this Software Product on to a hard disk or any other storage device of a computer, or loading any of the Components into the memory of any computer, constitutes use of the Software and shall acknowledge your acceptance of the terms and conditions of this License Agreement and your agreement to bound thereby.

1. GRANT OF LICENSE

Catalyst Development grants you as an individual, a personal, non-exclusive, non-transferable license to install the Software Product using an authorized serial number. If you are an entity, Catalyst grants you the right to appoint an individual within your organization to use and administer the Software Product subject to the same restrictions enforced on individual users. You may not network the Software or otherwise use it on more than one workstation or computer at the same time. Contact Catalyst for more information regarding multi-developer site licensing.

You may install the Software Product on one or more workstations or computers expressly for the purposes of evaluating the performance of the Software for a period of no more than thirty (30) days. If continued use of the Software is desired after the evaluation period has expired, then the Software Product must be purchased and/or registered with Catalyst Development for each computer or workstation. The Software Product must be removed from all unregistered workstations or computers after the evaluation period has expired.

2. COPYRIGHT

Except for the licenses granted by this agreement, all right, title, and interest in and to the Software Product (including, but not limited to, all copyrights in any executable programs, modules, controls, libraries, electronic documentation, text and example programs), any printed materials and copies of the Software Product are owned by Catalyst Development. The Software Product is protected by copyright laws and international treaty provisions. Therefore you must treat the Software Product like any other copyrighted material except that you may (i) make one copy of the Software solely for backup or archival purposes, or (ii) transfer the Software to a single hard disk, provided you keep the original solely for backup or archival purposes. You may not copy any printed materials that may accompany the Software Product. All rights not specifically granted in this Agreement, including Federal and International Copyrights, are reserved by Catalyst Development.

3. REDISTRIBUTION

(a) In addition to the rights granted in section 1, you are granted the right to use and modify those portions of the Software designated as "example code" for the sole purposes of designing, developing, and testing your software product, and to reproduce and distribute the example code, along with any modifications thereof, only in object code form, provided that you comply with section 3(c).

(b) In addition to the rights granted in section 1, you are granted a non-exclusive, royalty-free right to reproduce and distribute the object code version of any portion of the Software Product, along with any modifications thereof, in accordance with the above stated conditions.

(c) If you redistribute the sample code or redistributable components, you agree to: (i) distribute the redistributables in object code only, in conjunction with and as a part of a software application product developed by you which adds significant and primary functionality to the Software; (ii) not use Catalyst Development's name, logo, or trademarks to market your software application product; (iii) include a valid copyright notice on your software product ; (iv) indemnify, hold harmless, and defend Catalyst Development from and against any claims or lawsuits, including attorney's fees, that arise or result from the use or distribution of your software application product; (v) not permit further distribution of the redistributables by your end user.

4. UPGRADES

If this copy of the Software is an upgrade from an earlier version of the Software, you must possess a valid full license to a copy of an earlier version of the Software to install and/or use this upgrade copy. You may continue to use each earlier version copy of the Software to which this upgrade copy relates on your computer after you receive this upgrade copy, provided that, (i) the upgrade copy and the earlier version copy are installed and/or used on the same computer only and the earlier version copy is not installed and/or used on any other computer; (ii) you comply with the terms and conditions of the earlier version's end user license agreement with respect to the installation and/or use of such earlier version copy; (iii) the earlier version copy or any copies thereof on any computer are not transferred to another computer unless all copies of this upgrade copy on such computer are also transferred to such other computer; and (iv) you acknowledge and agree that any obligation Catalyst may have to support and/or offer support for the earlier version of the Software may be ended upon availability of the upgrade.

5. LICENSE RESTRICTIONS

You may not rent, lease or transfer the Software. You may not reverse engineer, decompile or disassemble the Software, except to the extent applicable law expressly prohibits the foregoing restriction. You may not alter the contents of a hard drive or computer system to enable the use of the evaluation version of the Software for an aggregate period in excess of the evaluation period for one license. Without prejudice to any other rights, Catalyst Development may terminate this License Agreement if you fail to comply with the terms and conditions of the agreement. In such event, you must destroy all copies of the Software Product.

6. CONFIDENTIALITY

(a) The Software contains information or material which is proprietary to Catalyst Development ("Confidential Information"), which is not generally known other than by Catalyst, and which you may obtain knowledge of through, or as a result of the relationship established hereunder with Catalyst. Without limiting the generality of the foregoing, Confidential Information includes, but is not limited to, the following types of information, and other information of a similar nature (whether or not reduced to writing or still in development): designs, concepts, ideas, inventions, specifications, techniques, discoveries, models, data, object code, documentation, diagrams, flow charts, research, development, methodology, processes, procedures, know-how, new product or new technology information, strategies and development plans (including prospective trade names or trademarks).

(b) Such Confidential Information has been developed and obtained by Catalyst by the investment of significant time, effort and expense, and provides Catalyst with a significant competitive advantage in its business.

(c) You agree that you shall not make use of the Confidential Information for your own benefit or for the benefit of any person or entity other than Catalyst, except for the expressed purposes described in this section, in accordance with the provisions of this Agreement, and not for any other purpose.

(d) You agree to hold in confidence, and not to disclose or reveal to any person or entity, the Software, other related documentation, your product Serial Number or any other Confidential Information concerning the Software other than to such persons as Catalyst shall have specifically agreed in writing to utilize the Software for the furtherance of the expressed purposes described in this section, in accordance with the provisions of this Agreement, and not for any other purpose.

(e) You acknowledge the purpose of this section is to protect Catalyst Development's ability to limit the use of the data and the Software generally to licensees, and to prevent use of Confidential Information concerning the Software by other developers or vendors of software.

7. CONTINUATION OF SERVICE

Some features of the Software may require the use of remote servers under the control of Catalyst Development to provide specific services. Catalyst makes no warranty as to the availability of these services and reserves the right to discontinue these services at any time and without warning. These services may only be accessed using the Application Programming Interfaces (API) provided by the Software Product and access is limited to licensees and evaluation users of the Software.

We may suspend or terminate your access to these services without liability if (i) we reasonably believe that the services are being used (or have been or will be used) in violation of the Agreement, (ii) we reasonably believe that suspending or terminating your access is necessary to protect our network or our other customers, or (iii) the suspension or termination is required by law. We will give you reasonable advance notice of suspension or termination under this section and a chance to cure the grounds on which the suspension or termination is based, unless we determine, in our reasonable commercial judgment, that an immediate suspension or termination is necessary to protect Catalyst or its other customers from imminent and significant operational or security risk.

8. LIMITED WARRANTY

If within thirty days of your purchase of this software product, you become dissatisfied with the Software for any reason, you may return the software to Catalyst Development (or your dealer, if you did not purchase it directly from Catalyst) for a refund of your purchase price. To return the Software, you must contact Catalyst Development and obtain a Return Material Authorization (RMA) number. Catalyst will not accept returns of opened or installed software without an RMA number. Returns may be subject to the deduction from your purchase price of a restocking fee and all shipping costs.

CATALYST PROVIDES NO REMEDIES OR WARRANTIES, WHETHER EXPRESS OR IMPLIED, FOR ANY SAMPLE APPLICATION CODE, TRIAL VERSION AND THE NOT FOR RESALE VERSION OF THE SOFTWARE. ANY SAMPLE APPLICATION CODE, TRIAL VERSION AND THE NOT FOR RESALE VERSION OF THE SOFTWARE ARE PROVIDED "AS IS".

CATALYST DISCLAIMS ALL OTHER WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTY OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH RESPECT TO THE SOFTWARE, THE ACCOMPANYING WRITTEN MATERIALS, AND ANY ACCOMPANYING HARDWARE.

9. LIMITATION OF LIABILITY

IN NO EVENT SHALL CATALYST OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITH LIMITATION, INCIDENTAL, CONSEQUENTIAL, SPECIAL, OR EXEMPLARY DAMAGES OR LOST PROFITS, BUSINESS INTERRUPTION, OR OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OR INABILITY OF THIS CATALYST PRODUCT, EVEN IF CATALYST HAS BEEN ADVISED OF SUCH DAMAGES.

APART FROM THE FOREGOING LIMITED WARRANTY, THE SOFTWARE PROGRAMS ARE PROVIDED "AS-IS", WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED. THE ENTIRE RISK AS TO THE PERFORMANCE OF THE PROGRAMS IS WITH THE PURCHASER. CATALYST DOES NOT WARRANT THAT THE OPERATION OF THE PROGRAMS WILL BE UNINTERRUPTED OR ERROR-FREE. CATALYST ASSUMES NO RESPONSIBILITY OR LIABILITY OF ANY KIND FOR ERRORS IN THE PROGRAMS OR DOCUMENTATION, OF/FOR THE CONSEQUENCES OF ANY SUCH ERRORS. THE LAWS OF THE STATE OF CALIFORNIA GOVERN THIS AGREEMENT.

10. GOVERNMENT-RESTRICTED RIGHTS

United States Government Restricted Rights. The Software and related documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software - Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer for such purposes is Catalyst Development Corporation, 56925 Yucca Trail #254, Yucca Valley, CA 92284

11. EXPORT CONTROLS

You agree to comply with all relevant regulations, including but not limited to those, of the United States Department of Commerce and with the United States Export Administration Act to insure that the Software is not exported in violation of United States law. You acknowledge that the Software is subject to export regulations and agree that you will not export, re-export, import or transfer the software in violation of any United States or other applicable laws, whether directly or indirectly, and you will not assist or facilitate others in doing so. You acknowledge that you have the responsibility to obtain any export classifications and licenses as may be required to comply with such laws.

12. PROHIBITED DESTINATIONS

The exportation, re-exportation, sale or supply of Catalyst products, software components or documentation, directly or indirectly, from the United States or by a United States citizen wherever located, to Cuba, Iran, North Korea, Sudan, Syria, or any other country to which the United States has embargoed goods, is strictly prohibited without prior authorization by the United States Government. You represent and warrant that neither the United States Bureau of Export Administration nor any other federal agency has suspended, revoked or denied your export privileges. Catalyst products, software components or documentation may not be exported or re-exported to anyone on the United States Treasury Department's list of Specially Designated Nationals or the United States Department of Commerce Denied Person's List or Entity List.

13. GOVERNING LAW

This License is governed by the laws of the State of California, without reference to conflict of laws principles. Any controversy or claim arising out of or relating to this contract, or the breach thereof, shall be settled by arbitration administered by the American Arbitration Association (“AAA”) under its Commercial Arbitration Rules, and judgment on the award rendered by the arbitrator(s) may be entered in any court having jurisdiction thereof. The arbitrator shall be a retired judge or attorney with at least 15 years commercial law experience and shall be selected either by mutual agreement of the parties or by AAA’s selection process. The parties shall be entitled to take discovery in accordance with the provisions of the California Code of Civil Procedure, including but not limited to CCP §1283.05. The arbitration shall be held in San Bernardino, California and in rendering the award the arbitrator must apply the substantive law of the State of California.

14. GENERAL PROVISIONS

This License Agreement contains the complete agreement between the parties with respect to the subject matter hereof, and supersedes all prior or contemporaneous agreements or understandings, whether oral or written. You agree that any varying or additional terms contained in any purchase order or other written notification or document issued by you in relation to the Software licensed hereunder shall be of no effect. The failure or delay of Catalyst to exercise any of its rights under this Agreement or upon any breach of this Agreement shall not be deemed a waiver of those rights or of the breach.

If any provision of this agreement shall be held by a court of competent jurisdiction to be contrary to law, that provision will be enforced to the maximum extent permissible, and the remaining provisions of this agreement will remain in full force and effect.

SocketTools and other trademarks contained in the Software are trademarks or registered trademarks of Catalyst Development Corporation in the United States and/or other countries. Third party trademarks, trade names, product names and logos may be the trademarks or registered trademarks of their respective owners. You may not remove or alter any trademark, trade names, product names, logo, copyright or other proprietary notices, legends, symbols or labels in the Software.

Copyright Information

Features Overview

The SocketTools .NET components can be used with any Visual Studio .NET programming language. SocketTools includes assemblies which can target every version of the .NET Framework, from version 1.1 through version 4.8.

Features of the SocketTools .NET Edition include:

- The SocketTools classes provide a simple interface that is easy to use and understand. There are no complicated methods or data structures to use, and most methods are overloaded to provide reasonable defaults appropriate for most applications. Most complex operations can be performed with only a few lines of code.
- There are no external dependencies on third party libraries or components. The SocketTools classes are managed code classes and are not "wrappers" around ActiveX controls or COM libraries.
- The class interfaces were designed to be as similar as possible to the COM version of the components, reducing the learning curve for developers who are already familiar with SocketTools and reducing the amount of time required to port an application to the .NET platform.
- A comprehensive design which supports both high-level operations as well as lower-level methods at the protocol level. For example, the File Transfer Protocol component has methods such as PutFile and GetFile which allow an application to easily upload and download files in a single method call. It also includes lower-level methods like OpenFile to open a file on the server and access it in a fashion similar to traditional file I/O operations.
- Support for both synchronous (blocking) and asynchronous (non-blocking) operation depending on the needs of the application. Asynchronous operation is supported by an event-driven model where the application is notified of networking events by events generated by the component. Event notification can be enabled, disabled and resumed completely under the control of the application, giving developers complete freedom in controlling their behavior of their software. Synchronous operation is also fully supported, enabling developers to easily write programs using a procedural programming style without the inherent complexity of an event-driven model.
- A thread-safe implementation that allows the classes to be easily used in a multithreaded application. SocketTools handles any internal synchronization required, ensuring that multiple, simultaneous operations can be performed safely and efficiently without extensive coding on the part of the developer.
- The SocketTools .NET Edition enables applications to take advantage of security features, including support for TLS 1.2 and AES 256-bit encryption without requiring any knowledge of data encryption or certificate validation. The components use the Windows CryptoAPI to provide security services, which means that there are no third-party security libraries that must be installed by your users. Taking advantage of the security features in the SocketTools .NET Edition is as simple as setting a few properties before connecting to the server. The protocol negotiation, data encryption and decryption is handled transparently by the control. From the perspective of the application developer, it is just as if it were a standard connection to the server.

The SocketTools .NET Edition includes everything professional software developers need to create complex programs, enabling developers to focus on their core application technology rather than the details of how a particular application protocol is implemented or understanding the specifics of network programming.

Getting Started

SocketTools is a large collection of components that can be used to create a variety of applications, so deciding what protocols and controls you'll need to use will be the first step. SocketTools covers several general categories, and there is some cross-over between the components in terms of functionality. We'll cover the most common programming needs and discuss what protocols should be used. Note that this section doesn't cover all of the controls in SocketTools, and more specific information for each component is available within the technical reference documentation.

One thing you'll discover as you start to use SocketTools is that the interface was intentionally designed to be consistent between many of the controls. For example, both the File Transfer Protocol and Hypertext Transfer Protocol controls can be used to upload and download files, and the properties, methods and events for both of those controls are very similar. Once you've become comfortable working with one of the controls, you'll find it very easy to use the other, related controls.

File Transfers

- File Transfer Protocol
- Hypertext Transfer Protocol

One of the most common requirements for an application is the ability to upload and download files, either over the Internet or between systems on a local intranet. There are two core protocols which are used for file transfers, the File Transfer Protocol (FTP) and the Hypertext Transfer Protocol (HTTP). The decision as to which protocol to use largely depends on whether or not the program must also perform any type of file management on the server. Because many of the methods in the FTP and HTTP components are similar, you may wish to use both and simply give your users an option as to which protocol they prefer to use.

If your program needs to upload files or manage the files on the server, we recommend that you use FTP. In addition to uploading and downloading files, FTP can be used to rename or delete files, create directories, list the files in a directory and perform a variety of other functions. On the other hand, if you primarily need to just download files, HTTP can be a better choice. The protocol is simpler and you're less likely to encounter some of the issues that can arise when using FTP from behind a firewall.

It is also an option to use FTP to upload and manage files and HTTP to download files within the same program. The important thing to keep in mind is that if you want to use HTTP and need to upload files, you must make sure that the server has been configured for it. Most web servers do not support the ability to upload files by default; it requires the administrator to specifically enable that functionality.

World Wide Web

- Hypertext Transfer Protocol

If you need to access documents or execute scripts on a web server, you'll want to use the Hypertext Transfer Protocol (HTTP) control. You can use the control to download files and post data to scripts. The control also supports the ability to upload files, either using the PUT command or by using the POST command, which is the same method used when selecting a file to upload using a form. The control can also be used to execute custom commands, allowing your application to take advantage of features like WebDAV, a distributed authoring extension to HTTP.

Electronic Mail

- Domain Name Services Protocol
- Internet Message Access Protocol

- Mail Message Class
- Post Office Protocol
- Simple Mail Transfer Protocol

There are a number of SocketTools components which can be used by an application that needs to send email messages or retrieve them from a user's mailbox. The email related controls can be broken into three groups, those that deal primarily with managing and retrieving messages for a user, those which are used to send messages and those which can be used for either purpose.

The two principal protocols used to manage a user's email are the Post Office Protocol (POP3) and the Internet Message Access Protocol (IMAP). POP3 is the protocol that the majority of Internet Service Providers (ISP) use to give their customers access to their messages. It is primarily designed to enable an application to download the messages from the mail server and store them on the local system. Once all of the messages have been downloaded, they are deleted from the server. The user's mailbox is essentially treated as a temporary storage area.

On the other hand, IMAP is designed to allow the application to manage the messages on the server. You can create new mailboxes, move messages between mailboxes and search for messages. Because IMAP can be used to access specific parts of a message, it's not necessary to download the entire message if you just want to read a specific part of it. In terms of the SocketTools controls, it's useful to think of the properties, methods and events in the IMAP control as a superset of those in the POP3 control. You'll find that methods used for accessing messages are very similar, but the IMAP component contains additional methods for managing mailboxes and performing operations that are specific to that protocol, such as the ability to search for messages.

To send an email message to someone, the protocol that you'll use is the Simple Mail Transfer Protocol (SMTP). The SocketTools control supports the standard implementation of this protocol, along with many of the extensions that have been added since its original design. Extended SMTP (ESMTP) provides features such as authentication, delivery status notification, secure connections using SSL/TLS and so on. Another component that you may use is the Domain Name Services (DNS) control, which your application can use to determine what servers are responsible for accepting mail for a particular user.

Common to both sending and receiving email messages is the need to be able to create and process those messages. An email message has a specific structure which is defined by a number of standards, collectively called the Multipurpose Internet Mail Extensions (MIME). The SocketTools Mail Message control can be used to create messages in the format, as well as parse existing messages so that you application can access the specific information that it needs. For example, you can use this component to attach files to a message as well as extract a specific file attachment from a message and store it on the local system.

Terminal Sessions

- Rlogin Protocol
- Telnet Protocol
- Terminal Emulation

If you need to establish an interactive terminal session with a server, there are two protocols that you can use. The most common is the Telnet Protocol; however, there is also the Rlogin protocol which is part of the Remote Command control. Either of these protocols are typically used in conjunction with the Terminal Emulation control, which provides ANSI and DEC VT-220 terminal emulation functionality. Used together, the user can login and interact with the server in the same way that they would use a console or character based terminal.

Newsgroups

- File Encoding Class
- Mail Message Class
- Network News Transfer Protocol

If you need to access newsgroups, the Network News Transfer Protocol will enable you to connect, list, retrieve and post articles. Because news articles have a format that is very similar to email messages, the Mail Message control can be used to parse articles that you've downloaded or create new articles to be posted. If you need to attach a file to the article that you're posting, the File Encoding control can be used to encode the file using the yEnc encoding algorithm, which has become the de facto standard on USENET.

General Concepts

This section of the developer's guide will cover the core networking protocols along with the general concepts related to Internet programming. Although it is not necessary to understand the lower level details of network programming in order to use SocketTools, it is useful to be familiar with the basic concepts and terminology.

- Windows Sockets
- Networking Protocols
- Application Protocols
- Domain Names
- Service Ports
- Client-Server Applications
- Client Sessions
- Secure Networking
- Digital Certificates

Windows Sockets

The Windows Sockets application program interface (API) specification was created by a group of companies, including Microsoft, in an effort to standardize the TCP/IP suite of protocols for the Windows operating system. Prior to Windows Sockets, each vendor developed their own proprietary libraries, and although they all had similar functionality, the differences were significant enough to cause problems for the software developers that used them. The biggest limitation was that, upon choosing to develop against a specific vendor's library, the developer was "locked" into that particular implementation. A program written against one vendor's product would not work with another's. Windows Sockets was offered as a solution, leaving developers and their end-users free to choose any vendor's implementation with the assurance that the product will continue to work.

There are two general approaches that you can take when creating a program that uses Windows Sockets. One is to code directly against the API, which requires learning how to make direct API calls to the system and how the Interop features of the .NET framework function. The other is to use a component (class library) like SocketTools which provides a higher-level interface by setting properties, calling methods and responding to events. This can provide a more natural programming interface, and it allows you to avoid much of the error-prone drudgery commonly associated with sockets programming. By simply referencing SocketTools in a project, setting some properties, calling a few methods and responding to events, you can quickly and easily write an Internet-enabled application.

SocketTools .NET provides a comprehensive interface to the networking subsystem that is powerful and flexible enough to build virtually any kind of application, yet doesn't have a steep learning curve for those developers who aren't experienced network programmers. In general, we believe that you'll be able to use SocketTools to write software for the Internet faster and easier than the native socket class that's part of the .NET framework.

Networking Protocols

Throughout the Developer's Guide, you will see the word "protocols" mentioned. There are two general types of protocols that will be discussed. The first type of protocol will be referred to as networking protocols. They are lower level protocols which define how data is exchanged between two systems. The two networking protocols that will be discussed are the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP).

There are also application protocols, which use the networking protocols to communicate. Application protocols deal with a specific type of functionality. For example, the File Transfer Protocol (FTP) is used to upload and download files, while the Simple Mail Transfer Protocol (SMTP) is used to send email messages. Conceptually, you can think of the networking protocols as defining the rules for how programs can communicate with one another over the Internet. The application protocols operate at a higher level, defining the rules for how a specific kind of task can be carried out, such as transferring a file from one computer to another.

- Transmission Control Protocol
- User Datagram Protocol

Transmission Control Protocol

When two computers wish to exchange information over a network, there are several components that must be in place before the data can actually be sent and received. Of course, the physical hardware must exist, which is typically either a network interface card (NIC) or a serial communications port for dial-up networking connections. Beyond this physical connection, however, computers also need to use a protocol which defines the parameters of the communication between them. In short, a protocol defines the "rules of the road" that each computer must follow so that all of the systems in the network can exchange data. One of the most popular protocols in use today is TCP/IP, which stands for Transmission Control Protocol/Internet Protocol.

By convention, TCP/IP is used to refer to a suite of protocols, all based on the Internet Protocol (IP). Unlike a single local network, where every system is directly connected to each other, an internet is a collection of networks, combined into a single, virtual network. The Internet Protocol provides the means by which any system on any network can communicate with another as easily as if they were on the same physical network. Each system, commonly referred to as a host, is assigned a numeric value which can be used to identify it over the network. These numeric values are known as IP addresses, and are usually represented as a string value that contains a series of numbers.

There are two versions of TCP/IP and two different IP address formats based on which version of the protocol is being used. For Internet Protocol v4 (IPv4), addresses are 32 bits wide and are represented by a sequence of four 8-bit numbers separated by periods. This is called dot-notation and looks something like **192.168.19.64**. This is the address format that many developers are familiar with because IPv4 continues to be the most commonly used version of the protocol. Internet Protocol v6 (IPv6) is the next generation of IP and it supports a much larger address space as well as a number of other features. IPv6 addresses are 128 bits wide and represented by a sequence of hexadecimal values separated by colons. As expected, this format is much longer than the simple dot-notation used by IPv4 address. A typical IPv6 address will look something like **fd7c:2f6a:4f4f:ba34::a32**, although there are certain shorthand notations that can be used. SocketTools supports both IPv4 and IPv6, and can automatically determine which version of the protocol should be used based on the address. Because IPv4 is still widely used, if given a choice between using IPv4 or IPv6, the SocketTools components will choose IPv4 for backwards compatibility whenever possible. However, an application can choose to exclusively use IPv6 if required.

When a system sends data over the network using the Internet Protocol, it is sent in discrete units called datagrams, also commonly referred to as packets. A datagram consists of a header followed by application-defined data. The header contains the addressing information which is used to deliver the datagram to its destination, much like an envelope is used to address and contain postal mail. And like postal mail, there is no guarantee that a datagram will actually arrive at its destination. In fact, datagrams may be lost, duplicated or delivered out of order during their travels over the network. Needless to say, this kind of unreliability can cause a lot of problems for software developers. What's really needed is a reliable, straightforward way to exchange data without having to worry about lost packets or jumbled data.

To fill this need, the Transmission Control Protocol (TCP) was developed. Built on top of IP, TCP offers a reliable, full-duplex byte stream which may be read and written to in a fashion similar to reading and writing a file. The advantages to this are obvious: the application programmer doesn't need to write code to handle dropped or out-of-order datagrams, and instead can focus on the application itself. And because the data is presented as a stream of bytes, existing code can be easily adopted and modified to use TCP.

TCP is known as a connection-oriented protocol. In other words, before two programs can begin to exchange data they must establish a "connection" with each other. This is done with a three-way handshake in which both sides exchange packets and establish the initial packet sequence numbers (the sequence number is important because, as mentioned above, datagrams can arrive out of order; this number is used to ensure that data is received in the order that it was sent). When establishing a connection, one program must assume the role of the client, and the other the server. The client is responsible for initiating the connection, while the server's responsibility is to wait, listen and respond to incoming connections. Once the connection has been established, both sides may send and receive data until the connection is closed.

User Datagram Protocol

Unlike TCP, the User Datagram Protocol (UDP) does not present data as a stream of bytes, nor does it require that you establish a connection with another program in order to exchange information. Data is exchanged in discrete units called datagrams, which are similar to IP datagrams. In fact, the only features that UDP offers over raw IP datagrams are port numbers and an optional checksum.

UDP is sometimes referred to as an unreliable protocol because when a program sends a UDP datagram over the network, there is no way for it to know that it actually arrived at its destination. This means that the sender and receiver must typically implement their own application protocol on top of UDP. Much of the work that TCP does transparently (such as generating checksums, acknowledging the receipt of packets, retransmitting lost packets and so on) must be performed by the application itself.

With the limitations of UDP, you might wonder why it's used at all. UDP has the advantage over TCP in two critical areas: speed and packet overhead. Because TCP is a reliable protocol, it goes through great lengths to ensure that data arrives at its destination intact, and as a result it exchanges a fairly high number of packets over the network. UDP doesn't have this overhead, and is considerably faster than TCP. In those situations where speed is paramount, or the number of packets sent over the network must be kept to a minimum, UDP is an appropriate protocol to use.

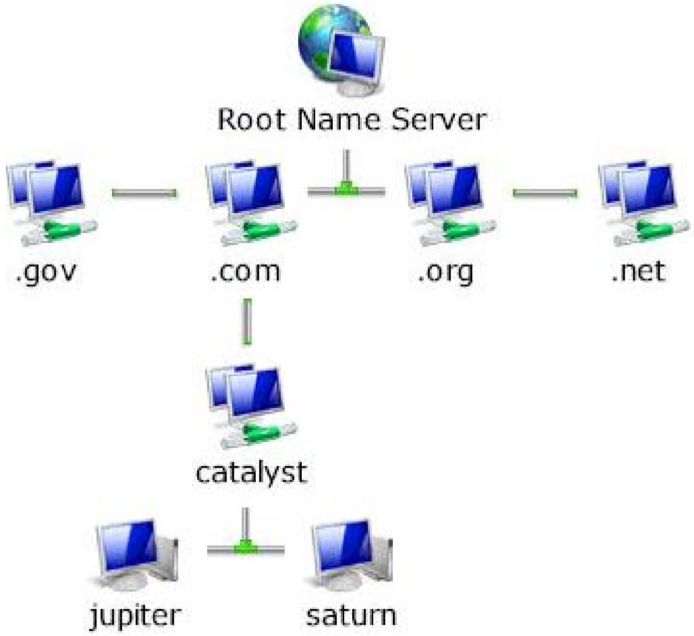
Application Protocols

The Internet application protocols are defined in standards documents called RFCs (Request For Comments) which are maintained by the Internet Engineering Task Force. The following protocols standards are implemented by the SocketTools components:

Document	Description
RFC 792	Internet Control Message Protocol
RFC 822	Standard for the Format of ARPA Internet Text Messages
RFC 854	Telnet Protocol Specification
RFC 868	Time Protocol
RFC 954	Nicname/Whois Protocol
RFC 959	File Transfer Protocol (FTP)
RFC 977	Network News Transfer Protocol
RFC 1034	Domain Name Services
RFC 1055	Serial Line IP (SLIP)
RFC 1282	Rlogin
RFC 1288	Finger User Information Protocol
RFC 1579	Firewall-Friendly FTP
RFC 1661	The Point-to-Point Protocol (PPP)
RFC 1738	Uniform Resource Locators
RFC 1869	SMTP Service Extensions
RFC 1939	Post Office Protocol Version 3
RFC 1945	Hypertext Transfer Protocol 1.0
RFC 1951	Deflate Compressed Data Format Specification
RFC 2045	Multipurpose Internet Mail Extensions (Part One)
RFC 2046	Multipurpose Internet Mail Extensions (Part Two)
RFC 2047	Multipurpose Internet Mail Extensions (Part Three)
RFC 2048	Multipurpose Internet Mail Extensions (Part Four)
RFC 2228	FTP Security Extensions
RFC 2616	Hypertext Transfer Protocol 1.1
RFC 2821	Simple Mail Transfer Protocol (SMTP)
RFC 2980	Common NNTP Extensions
RFC 3501	Internet Message Access Protocol Version 4

Domain Name System

An application must have several pieces of information to exchange data with a program running on another system. The first is the Internet Protocol (IP) address of the computer system on which the other program is running. Although this address is internally represented by a numeric value (either 32 or 127 bits wide), it is typically identified by a logical name called a host name or fully qualified domain name. Host names are divided into several parts separated by periods, called domains. The structure is hierarchical, with the top-level domains defining the type of organization that network belongs to, and sub-domains further identifying the specific network. Everyone who has used a web browser is familiar with host names such as **www.microsoft.com**.



In this figure, the top-level domains are "gov" (government agencies), "com" (commercial organizations), "edu" (educational institutions) and "net" (Internet service providers). The fully qualified domain name is specified by naming the host and each parent sub-domain above it, separating them with periods. For example, the fully qualified domain name for the "jupiter" host would be "jupiter.sockettools.com". In other words, the system "jupiter" is part of the "catalyst" domain (a company's local network) which in turn is part of the "com" domain (a domain used by all commercial enterprises).

To use a host name instead of an IP address to identify a specific system or network, there must be some correlation between the two. This is accomplished by one of two means: a local host table or a name server. A host table is a text file that lists the IP address of a host, followed by the names by which it is known. A name server is a system which can be presented with a host name and will return that host's IP address. This approach is advantageous because the host information for the entire network is maintained in one centralized location, rather than being scattered over every system on the network.

The standard protocol used to convert a host name into an IP address is called the Domain Name Service (DNS) protocol. All of the SocketTools networking libraries have the ability to automatically convert between host names and IP addresses, and in most cases they can be used interchangeably. For example, those methods which require that you specify the name of a server to connect to, you can use either its host name or its IP address. In addition, SocketTools has a control that specifically supports the Domain Name Service protocol, enabling your application to send specialized queries to the name server. Later in the Developer's Guide there will be information about how DNS can be used in a number of different types of applications.

Service Ports

In addition to the IP address of the server, an application also needs to know how to address the specific program that it wishes to communicate with. This is accomplished by specifying a service port, a number between 1 and 65535 that uniquely identifies an application running on the system. A port can be referred to by its number, or by a name that is associated with that number. Like hostnames, service names are usually matched to port numbers through a local file, commonly called services. This file lists the logical service name, followed by the port number and protocol used by the server.

A number of standard service names are used by Internet-based applications and these are referred to as Well Known Services (WKS). These services are defined by a standards document and include common application protocols used for transferring files, accessing documents on a webserver or sending and receiving email messages. In most cases, when connecting to a service using the SocketTools libraries, they will default to the appropriate port number for that server. For example, the File Transfer Protocol control has default port values for standard and secure connections. Specifying a different port number is only necessary if you know that the server has been configured to use a non-standard port number.

It is important to remember that a service name or port number is a way to address an application running on a server. Because a particular service name is used, it doesn't guarantee that the service is available, just as dialing a telephone number doesn't guarantee that there is someone at home to answer the call.

Client-Server Applications

Programs written to use TCP are developed using the client-server model. As mentioned previously, when two programs wish to use TCP to exchange data, one of the programs must assume the role of the client, while the other must assume the role of the server. The client application initiates what is called an active open. It creates a socket and actively attempts to connect to a server program. On the other hand, the server application creates a socket and passively listens for incoming connections from clients, performing what is called a passive open. When the client initiates a connection, the server is notified that some process is attempting to connect with it. By accepting the connection, the server completes what is called a virtual circuit, a logical communications pathway between the two programs. It's important to note that the act of accepting a connection creates a new socket; the original socket remains unchanged so that it can continue to be used to listen for additional connections. When the server no longer wishes to listen for connections, it closes the original passive socket.

To review, there are five significant steps that a program which uses TCP must take to establish and complete a connection. The server side would follow these steps:

1. Create a socket.
2. Listen for incoming connections from clients.
3. Accept the client connection.
4. Send and receive information.
5. Close the socket when finished, terminating the conversation.

In the case of the client, these steps are followed:

1. Create a socket.
2. Specify the address and service port of the server program.
3. Establish the connection with the server.
4. Send and receive information.
5. Close the socket when finished, terminating the conversation.
6. Only steps two and three are different, depending on if it's a client or server application.

Client Sessions

One of the first issues that you'll encounter when developing your applications is the difference between blocking and non-blocking client sessions. Whenever you perform some network operation, it may not be able to complete immediately and return control back to your program. For example, a read on a socket cannot complete until some data has been sent by the server. If there is no data waiting to be read, one of two things can happen: the function can wait until some data has been written on the socket, or it can return immediately with an error that indicates that there is no data to be read.

The first case is called a synchronous or blocking session. In other words, the program is "blocked" until the request for data has been satisfied. When the server does write some data on the socket, the read operation will complete and execution of the program will resume. The second case is called an asynchronous or non-blocking session, and requires that the application recognize the error condition and handle the situation appropriately. Programs that use non-blocking sockets typically use one of two methods when sending and receiving data. The first method, called polling, is when the program periodically attempts to read or write data from the socket (typically using a timer). The second, and preferred method, is to use what is called asynchronous notification. This means that the program is notified whenever a socket event takes place, and in turn can respond to that event. For example, if the remote program writes some data to the socket, an event is generated so that program knows it can read the data from the socket at that point.

With Visual Studio .NET, it is recommended that most applications using blocking (synchronous) sessions. However, it should be noted that blocking sockets can introduce some special problems in single-threaded applications. The blocking function will allow the application to continue processing messages from the operating system. Because messages are being processed, this means that the program can be re-entered at a different point with the blocked operation suspended on the program's stack. For example, consider a program that attempts to read some data from the socket when a button is pressed. Because no data has been written yet, it blocks and the program begins processing system messages. The user then presses a different button, which causes code to be executed, which in turn attempts to read data from the socket, and so on.

To resolve the general problems with blocking sockets, the Windows Sockets standard states that there may only be one outstanding blocked call per thread of execution. This means that applications that are re-entered will encounter errors whenever they try to take some action while a blocking function is already in progress. It is recommended that worker threads be created to manage each client session, with each thread owning a specific instance of the class being used. This resolves the potential conflict between multiple blocking sessions in the same thread, and will improve the performance of the application overall.

The only time that non-blocking (asynchronous) sessions are recommended are with single-threaded applications with a user interface that the user must interact with as the network operation is being performed. It should be noted that there is additional overhead involved with asynchronous sessions which can negatively impact the overall performance of the application. When possible, a multithreaded solution using worker threads to handle the client session in the background is preferred.

In summary, there are three general approaches that can be taken when building an application with the control in regard to blocking or non-blocking sockets:

- Use a blocking (synchronous) session. In this mode, the program will not resume execution until the socket operation has completed. This is the recommended approach, offering better performance and simplified debugging.
- Use a non-blocking (asynchronous) session, which allows your application to respond to events. For example, when the remote system writes data to the socket, an OnRead event is generated for the control. This approach should only be used with single-threaded applications with a user interface. There is additional overhead for processing network messages and debugging can be more complex.

- Use a non-blocking (asynchronous) session and poll the socket for data using properties such as IsReadable and IsWritable. This approach incurs additional overhead and it is recommended that you create worker threads to handle a blocking client session rather than poll for data.

If you decide to use a non-blocking session in your application, it's important to keep in mind that you must check the return value from every read and write operation, since it is possible that you may not be able to send or receive all of the specified data. Frequently, developers encounter problems when they write a program that assumes a given number of bytes can always be written to, or read from, the socket. In many cases, the program works as expected when developed and tested on a local area network, but fails unpredictably when the program is released to a user that has a slower network connection (such as a serial dial-up connection to the Internet). By always checking the return values of these operations, you insure that your program will work correctly, regardless of the speed or configuration of the network.

Secure Networking

Security and privacy is a concern for everyone who uses the Internet, and the ability to provide secure transactions over the Internet has become one of the key requirements for many business applications. SocketTools .NET has the ability to establish secure, encrypted connections with servers using one of several standard security protocols. Although most of the technical issues such as data encryption are handled internally by the component itself, a general understanding of the standard security protocols is useful when designing your own applications.

When you establish a connection to a server over the Internet (for example, a web server), the data that you exchange is typically routed over dozens of computer systems until it reaches its destination. Any one of these systems may monitor and log the data that it forwards, and there is no way for either the sender or receiver of that data to know if this has been done. Exchanging information over the Internet could be likened to talking with someone in a public restaurant. Anyone can choose to listen to what you're saying, and unless they introduce themselves, you have no idea who they are or if they've even heard what you said.

To ensure that private information can be securely exchanged over the Internet, two basic requirements must be met: there must be a way to send that information so that only the sender and the receiver can understand what is being exchanged, and there must be a way for them to determine that they each are in fact who they claim to be. The solution to the first problem is to use encryption, where a key is used to encrypt and decrypt the data using a mathematical formula. The second problem is addressed by using digital certificates. These certificates are issued by a certificate authority (CA), which is a trusted third-party organization who verifies the individual or company which is issued a certificate are who they claim to be. These two concepts, encryption and digital certificates, are combined to provide the means to send and receive secure information over the Internet.

The Secure Sockets Layer (SSL) protocol was originally developed by Netscape as a way to exchange information securely over the Internet, and is no longer widely used. Improvements to SSL have resulted in the Transport Layer Security (TLS) protocol, and it has become the the standard for secure communications over the Internet. Both of these protocols are designed to allow a private exchange of encrypted data between the sender and receiver, making it unreadable by an intermediate system. Using the restaurant analogy, it would be as if two people were speaking in a language that only they could understand. Although someone sitting at the next table could listen in on the conversation, they wouldn't have any idea what was actually being said.

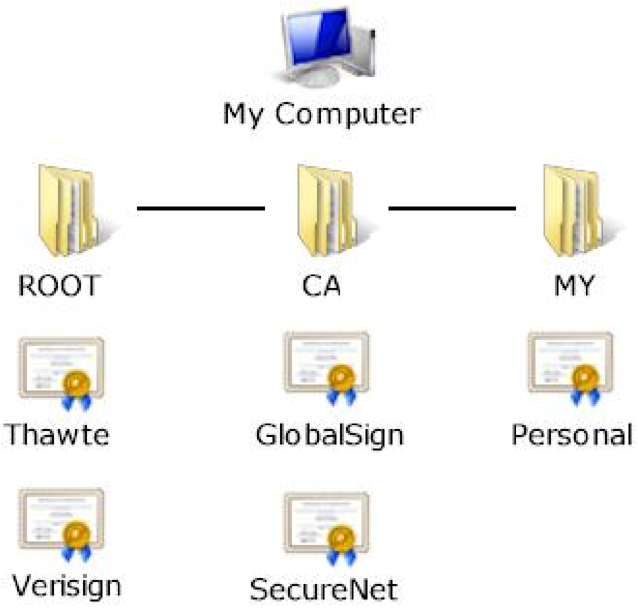
A secure connection, for example between a web browser and a server, begins with what is called the handshake phase where the client and server identify themselves. When the client first connects with the server it sends a block of data to the server and the server responds with its digital certificate, along with its public key and information about what type of encryption it would like to use. Next, the client generates a master key and sends this key to the server, which authenticates it. Once the client and server have completed this exchange, keys are generated which are used to encrypt and decrypt the data that is exchanged. With the handshake completed, a secure connection between the client and server is established. SocketTools handles the handshake phase of the secure connection automatically and does not require any additional programming. If a secure connection cannot be established, an error is returned and the network connection is closed.

After the handshake phase has completed, the client may choose to examine the digital certificate that has been returned by the server. The information contained in the certificate includes the date that it was issued, the date that it expires, information about the organization who issued the certificate (called the issuer) and to whom the certificate was issued (called the subject of the certificate). The client may also validate the status of the certificate, determining if it was issued by a trusted certificate authority and was returned by the same company or individual it was issued to. There may be certain cases where the client determines that there's a problem with the certificate (for example, if the certificate's common name does not match the domain name of the server), but chooses to continue communicating with the server. Note that the connection with the server will still be secure in this case. In other cases, for example if the certificate has expired, the client may choose to terminate the connection and warn the user.

Digital Certificates

With secure connections, digital certificates are used to exchange public keys for data encryption and to provide identification information. This information typically includes the organization that was issued the certificate, its physical location and so on. The certificate itself is used to validate that the public key actually belongs to the entity it was issued to. The certificate also includes information about the Certification Authority (CA) who issued the certificate. The CA is responsible for validating the information provided by that organization, and then digitally signing the certificate. This establishes a relationship between the two so that when others validate the certificate, they know that it has been issued by a trusted third-party. For example, let's say that a company wants to implement a secure site so people can order products online. They would provide information about their company (organizational contacts, financial information and so on) to a trusted third party organization such as Verisign or Thawte. That organization would then verify that the information they provided was complete and correct, and then would issue a signed certificate to them, which they install on their server. When a user connects to their server and checks the certificate, they see that it was issued by a trusted Certification Authority. In essence, the user is saying that because they trust the Certificate Authority, and the Certificate Authority trusts the company to whom the certificate was issued, they will trust the company as well.

To establish this relationship between the Certification Authority and the organization a certificate is issued to, there needs to be a root certificate which has been signed by the same trusted organization. This serves as the beginning of the certification path that is used to validate signed certificates. Using the above example, on the user's system there is a root certificate for Verisign, signed by Verisign. Root certificates are maintained in the local system's certificate store which is essentially a database of digital certificates. This database is structured so that different types of certificates can be organized in one central location on the system, and a standard interface is provided to enumerate and validate these certificates. Certificates are associated with a store name, allowing them to be easily categorized. For example, root certificates are stored under the name "Root", while a user's personal certificates (along with their private keys) are stored under the name "My".



When the Windows operating system is installed, there is a certificate store that contains the root certificates for the major Certification Authorities. However, there are situations where additional certificates may need to be added to the system. To facilitate this, there is a tool called CertMgr.exe which allows a user to install certificates, as well as export or remove certificates from the certificate store. When managing your system's certificate store, you should take the same care that you do when making changes to the system registry. Inadvertently removing a certificate could result in errors when attempting to access secure systems.

In general, the one situation where certificate management becomes important is when you want to develop your own secure server. This is because your server needs to have a signed certificate to send to the client in order to establish the secure connection. For general-purpose commercial applications, this generally means you would need to obtain a certificate that has been signed by a Certification Authority such as Verisign or Thawte. This certificate would then be installed in the certificate store on the server. However, for development purposes it may be inconvenient to purchase a certificate. There also may be situations in which an organization wishes to function as its own Certification Authority and issue certificates themselves. This allows the organization to control how certificates are managed and can be ideal for secure applications that are designed for the corporate intranet. A utility for creating self-signed root certificates and server certificates is included with SocketTools.

SocketTools Development

The SocketTools .NET Edition provides a comprehensive collection of managed code classes for performing a variety of Internet related programming tasks. Although the number of properties, methods and events may appear daunting, once you begin using SocketTools in your own applications you'll find that the various controls are designed to work together in a cohesive fashion. After you've familiarized yourself with one control, the others will become much simpler to use.

Throughout the Developer's Guide there are some general concepts and terminology used that are essential to understanding how SocketTools works. Each of these concepts is explored in detail, however a general, broad overview can also be useful when you are just getting started.

Protocols

A protocol, in terms of how the word is used in SocketTools, refers to the rules for how programs communicate with one another over a network. There are low level networking protocols such as TCP and UDP, as well as high level application protocols like FTP and HTTP. It can be helpful to think of a protocol as a sort of language; for two programs to communicate with each other, they must agree upon a protocol and understand how it is implemented.

Connections

The process of establishing a connection enables one program to communicate with another. Connection requests are made by client applications, and accepted by server applications. When the server accepts the connection request, the connection is completed. When you use the Connect method to successfully establish a connection to a server, a client session is created. SocketTools uses a one-to-one relationship between an instance of a control and a client session. By creating multiple instances of a control, an application can create multiple client/server sessions if necessary.

Sessions

A session refers to an active connection between a client and server program. This term is typically used interchangeably with connection; however in some cases a single session may involve multiple network connections. For example, the File Transfer Protocol control establishes one connection, called the command channel, when the client initially connects to the server. However, when a file is being uploaded or downloaded, a second connection called the data channel is created just for that transfer. When the transfer completes, the second connection is terminated while the original command channel connection remains active. Even though there are multiple connections being made, SocketTools considers it to be a single client session. An active session is referenced by the instance of the control that was used to create the session. When the session is no longer needed, the control's Disconnect method will terminate the connection to the server and release the resources allocated for that session. After that point, the session is no longer valid and subsequent function calls using the control cannot be made until another connection is established.

Authentication

Many servers require that clients authenticate themselves by providing user names and passwords. Different application protocols implement several different types of authentication, and some protocols may support more than one authentication method. SocketTools provides one of two general types of authentication methods, depending on the protocol. For protocols which require the client to authenticate itself, the controls will provide a Login method. For protocols where authentication is optional, the controls will provide an Authenticate method. Refer to the technical reference for the specific protocol to determine if authentication is required.

Events

Developers who use programming languages such as Visual Basic will find the concept of events and event handling to be very familiar. In general terms, the SocketTools documentation uses "event" to refer to a mechanism where the control notifies the application that an operation has completed, some action has taken place or a change in status has occurred. One example of an event is a connection event, which is generated whenever an asynchronous network connection is completed by the client. Another example is a progress event, which is generated periodically by the control to inform the client of its progress as it sends or receives data. To determine what events are available in a specific control, refer to the documentation. More specific information about event handling is provided later in this guide.

Application Design

The SocketTools .NET Edition is designed to be flexible enough to address the needs of developers who have very basic needs, as well as those who have more complex requirements. As a result, the properties and methods for a control can be broken down into two general categories: a high level interface to perform common tasks, and a lower level interface which provides more control at the expense of being somewhat more complicated and requiring more coding. For example, consider the Hypertext Transfer Protocol (HTTP) control which has a variety of high level methods such as GetFile, PostData and so on. Using these methods, your application can perform the most common tasks for that protocol with a minimum of coding. You don't need to even understand the basics of how the protocol works, or what the control is doing. The high level methods allow you to program against the control as though it is a "black box", where you can provide the input and process the output without concerning yourself with the details of what's going on behind the scenes.

However, in some cases it's necessary for an application to have more direct control over how the control operates or to take advantage of features that aren't explicitly supported by one of the higher level methods. As an example, the HTTP control also has methods like Command, which enable you to send custom commands to a web server. Normally, for operations like retrieving a file or posting data to a script, this isn't necessary. But if your application needs to use WebDAV, a set of extensions to the HTTP protocol to support distributed web authoring, then the lower level methods like Command enable you to do this.

If you are generally new to Internet programming or are just getting started with SocketTools, we recommend that you begin familiarizing yourself with the higher level methods using a basic synchronous (blocking) connection in a single-threaded application. Once you become more familiar with how the control works, then you can move on to more complex applications which leverage the lower level methods, taking advantage of asynchronous networking connections and so on.

One of the common pitfalls that developers can encounter with a large toolkit like SocketTools is the inclination to over-design the application from the start, and then become frustrated because they don't yet have a clear picture of how all the pieces fit together. Start out with a basic design and then as you become more familiar with how the SocketTools controls work, expand on it.

Program Structure

Applications which use the SocketTools classes will tend to have a similar structure, regardless of the specific protocol or programming language. While the details vary based on the control being used, the implementation can be broken down into several general steps:

- Initializing the class instance
- Connecting to the server
- Authenticating the client
- Performing one or more operations
- Disconnecting from the server
- Uninitializing the class instance

Initialization prepares an instance of the class to be used by your program, and is the first step that must be performed before you can use any other methods. Next, a connection is established with the server using the information provided by your program. For example, most of the connection methods require that you provide a host name, port number, a timeout period for synchronous operations and any additional options.

If the protocol requires that you authenticate the client in order to use the service, your application needs to provide this information. Once the client has been authenticated, it can then perform one or more operations, such as downloading a file, sending an email message and so on.

After you have finished, you disconnect from the server. Finally, before your program terminates, you uninitialize the class instance which causes it to perform any necessary housekeeping prior to releasing any system resources which were allocated on behalf of your program.

Class Initialization

When you begin developing your application using SocketTools .NET, the first thing that must happen is the class instance must be initialized. The initialization method serves two purposes. It loads the networking libraries required to establish a connection and it validates the runtime license key that you provide. The runtime license key is a string of characters which identifies your license to use and redistribute SocketTools. It is unique to your product serial number and must be used when redistributing your application to an end-user.

Creating an instance of the class with your runtime license key can be accomplished in one of two ways, depending on personal preferences and the design of your application. The simplest approach, and one familiar to developers who have used the SocketTools ActiveX control, is to explicitly call the Initialize method in your code. The other more advanced approach is to define the RuntimeLicense attribute for the assembly that is referencing the class. This attribute is set in the AssemblyInfo.vb or AssemblyInfo.cs module that is part of the project. For more information on specific usage, refer to the Technical Reference.

Developers who are evaluating SocketTools will not have a runtime license key and must pass an empty string to the **Initialize** method. This will enable that instance of the class to load on the development system during the evaluation period, but will prevent component from being redistributed to an end-user until a license has been purchased.

If you install the product with a serial number, the runtime license key will be automatically created for you during the installation process. If you have installed an evaluation copy of SocketTools and then purchased a license, the license key can be created using the License Manager utility that was included with the product. Simply select the License | Header File menu option and select the programming language that you are using.

The runtime license key is normally stored in the Include folder where you installed SocketTools and is defined in a file named SocketToolsLicense which can be included with your application. For example, C# programmers would use the SocketToolsLicense.cs header file while Visual Basic programmers would use the SocketToolsLicense.vb module. The Visual Basic module would define the runtime license key as:

```
'
' SocketTools 10.0
' Copyright 2023 Catalyst Development Corporation
' All rights reserved
'
' This file is licensed to you pursuant to the terms of the
' product license agreement included with the original software
' and is protected by copyright law and international treaties.
'
Public Const SocketToolsLicenseKey As String = ""
```

This could either be included with your Visual Basic.NET application or you could simply copy the string into your application. The class could then be initialized like this:

```
'
' Initialize the control using the specified runtime license key;
' if the key is not specified, the development license will be used
'
If Socket.Initialize(SocketToolsLicenseKey) = False Then
    MsgBox("Unable to initialize SocketTools component")
End If
```

A return value of **true** indicates that the class was initialized successfully. A return value of **false** indicates that the class could not be initialized with the specified runtime license key. The **LastError** property will contain the error code that indicates the exact cause of the error.

An application is only required to call the Initialize method once, but it must be called for each instance of the class that is created. It is safe to call the initialization method more than once, but for each time that it is called, you must call the Uninitialize method for that class before your program terminates. In other words, if you called Initialize at the beginning of your program, you must call Uninitialize before your program ends. The Uninitialize method performs any necessary housekeeping operations, such as returning memory allocated for the class back to the operating system. If there are any open connections at the time that the Uninitialize method is called, they will be aborted. After the class has been uninitialized, you must call the Initialize method again before setting any properties or calling any methods in that instance of the class.

Secure Connections

The SocketTools .NET Edition supports the ability to create secure connections using the standard SSL and TLS protocols. In most cases, it is as simple as setting the **Secure** property to **true** or specifying an additional option when the **Connect** method is called. In some cases, certain Internet application protocols have additional requirements in terms of how the secure connection is established . Secure connections may either be implicit or explicit, depending on the protocol. An implicit secure connection is one where the client and server begin negotiating the security options as soon as the connection is established. In most cases, a server which accepts secure implicit connections listens on a port number that is different from the default port it uses for standard, non-secure connections. An example of this is the Hypertext Transfer Protocol (HTTP) which accepts standard connections on port 80 and secure connections on port 443. When a client connects to port 443, the server automatically assumes that the client wants a secure connection.

On the other hand, an explicit connection requires that the client explicitly specify to the server that it wants a secure connection. Typically this is done by the client sending a command to the server that causes the server to begin negotiating with the client to establish a secure session. An example of this is the File Transfer Protocol (FTP), where the client can use the AUTH command to tell the server that it wants a secure connection. Servers may also support both explicit and implicit secure connections, based on which port the client connects to. SocketTools supports both implicit and explicit secure connections. If the **Secure** property is set to **true** prior to calling the **Connect** method, then an implicit secure connection is established. Setting the **Secure** property to **true** after a connection has been established will cause SocketTools to begin negotiating a secure connection at that time.

In addition to establishing a secure connection, you may also be required to provide additional authentication information to the server in form a client certificate. For example, a server may require that the client provide a certificate in addition to or instead of a username and password. To support this, your application must specify the security credentials for the client prior to establishing a connection. For more information, refer to the **CertificateStore** and **CertificateName** properties in the Technical Reference.

Network Input/Output

Each of the SocketTools networking classes provides methods for exchanging data between your application and the server. At the lowest level, this is done by calling the Write method for sending data and the Read method for receiving data. In most cases, these methods exchange data as a stream of bytes without any regard for the actual content. It is important to note that if the data being read or written is binary, it is recommended that applications pass byte arrays, not strings, to the Read and Write methods if possible.

When working at this very low level, it is important to understand how data is exchanged over the network. Many developers are inclined to think of the data that is sent or received in terms of discrete blocks, or packets. The expectation is that if they send a certain number of bytes of data in a single write, the server will receive that number of bytes in a single read. However, this is not how TCP works, and by extension, not how the SocketTools libraries work with regards to this kind of low level network I/O. The Transmission Control Protocol (TCP) is called a stream-oriented protocol because data is exchanged between the client and server as a stream of bytes. While TCP will guarantee that the data will arrive intact, with the bytes received in the same order that they were written, there is no guarantee that the amount of data received in a single read operation on the socket will match the amount of data written by the server.

For example, consider a server that sends data to a client in four separate operations, each containing 1024 bytes of data. While it is convenient to think of these as discrete blocks of data, TCP considers it to be a stream of 4096 bytes. The client may receive that data in a single read on the socket, returning all 4096 bytes. Alternatively, it may read the socket, and only receive the first 1460 bytes; subsequent reads may return another 1460 bytes, followed by the remaining 1176 bytes. Applications which make assumptions about the amount of data they can read or write in a single operation may work in some environments, such as on a local network, but fail on slower connections.

A general rule to use when designing an application using TCP is to consider how the program would handle the situation where reading n bytes of data only returns a single byte. If the application can correctly handle this kind of extreme case, then it should function correctly even under adverse network conditions.

In some situations it may be desirable to design the application to exchange information as discrete messages or blocks of data. While this isn't directly supported by TCP, it can be implemented on top of the data stream. There are several methods that can be used to accomplish this, depending on the requirements of the application:

1. Exchange the data as fixed length structures. This is the simplest approach, and has very little or no overhead. The client and server can either use predefined values, or negotiate the size of the data structures when the connection is established.
2. Prefix variable-length data structures with the number of bytes being sent. The length value could be expressed either as a native integer value, or as a fixed-length string that is converted to a numeric value by the application. This allows the receiver to read this fixed length value, and then use that value to determine how many additional bytes must be read to obtain the complete message or data structure.
3. Prefix the data with a unique byte or byte sequence that would normally not be found in the data stream. This would be followed by the data itself, with a complete message received when another unique byte sequence is encountered. Alternatively, a unique byte sequence could be used to terminate a message. This is the approach that many Internet application protocols use, such as FTP, SMTP and POP3. Commands are sent as one or more printable characters, terminated with a carriage-return (CR) and linefeed (LF) byte sequence that tells the server that a complete command has been received.
4. A combination of the above methods, using unique byte sequences, the message length

and even additional information such as a CRC-32 checksum or MD5 hash to validate the integrity of the data. This would effectively create an "envelope" around the data being exchanged, and additional checks could be made to ensure that the data has been received and processed correctly.

Regardless of the method used, for best performance it is recommended that the application buffer the data received and then process the data out of that buffer. When using asynchronous (non-blocking) connections, the application should read all of the data available on the socket, typically in a loop which adds the data to the buffer and exiting the loop when there is no more data available at that time.

It is important to keep in mind that all of this is only required if you decide to use the lower level methods in the SocketTools controls. The higher level methods automatically handle the lower level network I/O for you. For example, the GetData method in the File Transfer Protocol control will retrieve a file from the server and return the entire contents to your application in a single method call. When using the high level methods, the details of how the data is read and processed is handled by the control and no additional coding is required on your part.

Event Handling

Event notification provides a mechanism for SocketTools to inform the application of a change in the status of the current session. Events are generally divided into two general categories, asynchronous network events and status events.

Asynchronous network events occur when a non-blocking connection is established and a network event occurs, such as a connection completing or data arriving from the server. Status events are used to indicate a change in status, such as a blocking operation being canceled or the progress of an operation such as reading a stream of bytes from the socket and storing it in a file. Note that asynchronous network events require that the **Blocking** property be set to **false**. The following events can be generated when SocketTools is in non-blocking mode:

OnConnect

This event is generated whenever a connection to a server has completed. Unlike a blocking connection, when the component is in non-blocking mode, a successful call to the **Connect** method does not indicate that you are actually connected to the host. Instead, it means that the connection process has been started. Your application will not actually be connected until the **OnConnect** event fires.

OnDisconnect

This event is generated whenever the server closes its socket and terminates the connection with your application. Note that this event will not fire when you disconnect from the host by calling the **Disconnect** method; it only fires when the server closes its connection to you. It is also important to keep in mind that although the server has disconnected from you, there still may be data buffered on your local system, waiting to be read. If you are performing any low-level network I/O, your program should continue to call the **Read** method until it returns a value of zero, indicating that all of the available data has been read.

OnRead

This event is generated whenever the server sends data to your application. Once this event has fired, it will not be triggered again until you read at least one byte of data that has been sent to you. It is recommended that you attempt to read and buffer all of the data that is available to be read in the socket. When the **Read** method returns a value less than or equal to zero, you should exit the event handler.

OnWrite

This event is generated whenever there is enough memory available in the local send buffers to accommodate some data. It is generated immediately after a connection has completed, which tells your application that it may begin sending data to the server. It will also be generated if a call to the **Write** method fails with the error that it would cause the thread to block. In this case, when the socket is able to accept more data, the **OnWrite** event will fire.

An important consideration when it comes to event handling is that all asynchronous network events are level triggered. This means that once an event is fired, it will not be fired again until some action is taken by the application to handle the event. This is most commonly found with **OnRead** events, which are generated when the server sends data to your application, signaling to you that there is data available to be read. Even though the server may continue sending you more data, another **OnRead** event will not be generated until you read at least one byte of the data that has been sent to you. This is done to prevent the application from being flooded with event notifications. However, failure to handle an event can cause event notification to appear to stall. It is recommended that you do not do excessive processing in an event handler that would cause the thread to block or enter a message loop. This can have a significant negative effect on performance and can lead to unexpected behavior on the part of your application. Instead, it's recommended that you buffer the data that you receive and then process that data after exiting the event handler.

Status related events are different because they do not depend on the value of the **Blocking** property, and are not directly related to asynchronous network operations. The most typical status event is the **OnProgress** event, which is used to provide information to the application about the status of a blocking operation, such as downloading a file from the server using the **GetFile** method. The possible status events are:

OnCancel

This event is used by the class to indicate that a blocking network operation has been canceled by a call to the **Cancel** method. It is important to note that when the **Cancel** method is called, the blocking socket operation will not immediately fail. An internal flag is set which causes the blocking operation to exit with an error. This means that the application cannot cancel an operation and immediately perform some other blocking function. Instead it must allow the calling stack to unwind, returning back to the blocking operation before making any further function calls. The **OnCancel** event handler should only be used for notification purposes or updating the internal state of the application. It is not recommended that you perform any network operations inside this event handler.

OnError

This event is used by the class to indicate an error has occurred. This event is only generated when a method is called, never as the result of setting a property value. The **OnError** event handler should only be used for notification purposes or updating the internal state of the application. It is not recommended that you perform any network operations inside this event handler.

OnProgress

This event is used by the class to inform the application of the progress of a blocking operation, such as a file transfer. Note that in some cases, the class may not be able to determine the total amount of data to be transferred, which would prevent a percentage from being calculated. In this case, because the server is unable to specify the total size of the resource, the class will not be able to calculate a percentage. Instead, it will simply inform the program of the amount of data copied to the local host up to that point.

OnTimeout

This event is used by the class to indicate a blocking operation has timed-out. A timeout period is specified by setting the **Timeout** property to a value greater than zero. The **OnTimeout** event handler should only be used for notification purposes or updating the internal state of the application. It is not recommended that you perform any network operations inside this event handler.

Status events are typically used to update a user interface. For example, the **OnProgress** event may be used to update a **ProgressBar** control, or a warning message may be displayed if an **OnError** event occurs.

Error Handling

Error conditions can occur in one of two general circumstances, either when setting a property in the control or when calling a method. If the error occurs when setting a property, an exception will be generated which must be caught and handled by the application. Failure to do this will typically result in the program displaying an error message and then terminating. For example, in Visual Basic.NET, the Try..Catch statements can be used to establish an error handler.

Methods are a bit different in that errors can be handled in one of two ways. By default, when a method is called it will return a value that indicates success or failure. For those methods that return boolean values, a value of **true** indicates success and a value of **false** indicates failure. For methods that return numeric values, such as the **Read** and **Write** methods, a return value of zero or greater indicates success, and a return value of -1 indicates failure.

If you prefer to handle exceptions, rather than check return values for each method call, SocketTools has a property called **ThrowError**. If this property is set to **true**, when a method fails it will throw an exception that must be caught by the application. In that case, if an error occurs without there being an exception handler in place, the application will typically terminate. The **ErrorCode** property of the exception class will specify the error that generated the exception. Refer to the Technical Reference for more information about the exception class specific to the class library being used.

To determine the error code for the last error generated by that instance of the class, use the **LastError** property. To display a description of the error to the user, the **LastErrorString** property will can be used. This returns a string that describes the error which corresponds to the value of the **LastError** property. It is permitted to set the **LastError** property to a value of zero in order to clear the last error code. It is important to note that the last error code only has meaning if the previous method call indicates the operation has failed. If the previous operation was successful, the value of the last error code will be undefined and should not be used.

Debugging Facilities

The SocketTools .NET class includes a built-in facility for generating debugging output in the form of a log file that provides information about the internal functions that it is using and the data that is being exchanged between the client and server. This is commonly referred to in the documentation as generating a trace log or enabling function logging.

To provide logging functionality for your application, you must redistribute the **SocketTools10.TraceLog.dll** library along with the the class library. This library is what performs the actual logging and must be placed in the same folder where your application is installed. Note that you cannot add this library as a reference to your project. It is used internally by SocketTools and cannot be used directly by your program.

To create a trace log, your application must set the **TraceFile** property to the name of a file, the **TraceFlags** property to the level of logging desired and then set the **Trace** property to **true**. The default level of logging, zero, specifies that general information about the function calls being made will be logged. The most detailed logging is provided by specifying a level of four. In that case, all data exchanged between your application and the server is logged. This provides the most information, however it also generates the largest log files. To disable logging, set the **Trace** property to false.

There are two important things that you need to consider when enabling trace logging. The first is that the log file is always appended to, never overwritten by the control. This means that the files can grow to be very large, particularly with trace that includes all of the data sent and received by your application. You can use the standard file I/O functions in your language to manage the log file or even write your own data out to the file. Each time the file is written to, SocketTools will open the file, append the logging data and then close the file; it will never keep the file open between operations. This is important because if your application terminates abnormally, it ensures all of the logging data has been written and there are no open file handles being held by that instance of the class. However, this does incur additional overhead and can impact the performance of your application. When possible, it is recommended that you enable logging around the code that you feel may be part of the problem you're trying to resolve, and then disable logging when it is no longer required. Simply enabling logging at the beginning of your application can result in unnecessarily large log files.

If your application uses multiple instances of the class, it is only necessary to enable logging in one of them. Once enabled, all network operations in the current thread will be logged, regardless of which instance has enabled logging.

SocketTools Class Overview

The SocketTools .NET Edition includes components that implement fourteen standard Internet application protocols, as well as components which provide support for general TCP/IP networking services, encoding and compressing files, processing email messages and ANSI terminal emulation. The following classes are included in the SocketTools .NET Edition:

SocketTools.DnsClient Class
The Domain Name Service (DNS) protocol is what applications use to resolve domain names into Internet addresses, as well as provide other information about a domain, such as the name of the mail servers which are responsible for receiving email for users in that domain. This class enables an application to query one or more nameservers directly, without depending on the configuration of the client system.

SocketTools.FileEncoder Class
The File Encoding class provides methods for encoding and decoding binary files, typically attachments to email messages. The process of encoding converts the contents of a binary file to printable text. Decoding reverses the process, converting a previously encoded text file back into a binary file. The class supports a number of different encoding methods, including support for the base64, uucode, quoted-printable and yEnc algorithms. The class can also be used to compress and expand data in a user-supplied buffer or in a file.

SocketTools.FtpClient Class
The File Transfer Protocol (FTP) class provides methods for uploading and downloading files from a server, as well as a variety of remote file management methods. In addition to file transfers, an application can create, rename and delete files and directories, list files and search for files using wildcards. The class provides both high level methods, such as the ability to transfer multiple files in a single method call, as well as access to lower level remote file I/O methods.

SocketTools.FtpServer Class
The File Transfer Server class provides an interface for implementing an embedded, lightweight server that can be used to exchange files with a client using the standard File Transfer Protocol. The server can accept connections from any third-party application or a program developed using the SocketTools.FtpClient class. The server supports active and passive mode file transfers, has compatibility options for NAT router and firewall support, and provides support for secure file transfers using explicit SSL/TLS sessions. Secure connections require that a valid SSL certificate be installed on the system.

SocketTools.HttpClient Class
The Hypertext Transfer Protocol (HTTP) class provides an interface for accessing documents and other types of files on a server. In some ways it is similar to the File Transfer Protocol in that it can be used to upload and download files; however, the protocol has expanded to also support remote file management, script execution and distributed authoring over the World Wide Web. The SocketTools Hypertext Transfer Protocol class implements version 0.9, 1.0 and 1.1 of the protocol, including features such as support for proxy servers, persistent connections, user-defined header fields and chunked data.

SocketTools.HttpServer Class
The Hypertext Transfer Server class provides an interface for implementing an embedded, lightweight server that can be used to provide access to documents and other resources using the Hypertext Transfer Protocol. The server can accept connections from any standard web browser, third-party applications or programs developed using the SocketTools.HttpClient class. The server includes support for CGI scripting, virtual hosting, client authentication and the creation of virtual directories and files. The server also supports secure connections using SSL/TLS. Secure connections require that a valid SSL certificate be installed on the system.

SocketTools.IcmpClient Class
The Internet Control Message Protocol (ICMP) is commonly used to determine if a server is reachable and

how packets of data are routed to that system. Users are most familiar with this protocol as it is implemented in the ping and tracert command line utilities. The ping command is used to check if a system is reachable and the amount of time that it takes for a packet of data to make a round trip from the local system, to the server and then back again. The tracert command is used to trace the route that a packet of data takes from the local system to the server, and can be used to identify potential problems with overall throughput and latency. The class can be used to build in this type of functionality in your own applications, giving you the ability to send and receive ICMP echo datagrams in order to perform your own analysis.

SocketTools.ImapClient Class

The Internet Message Access Protocol (IMAP) is an application protocol which is used to access a user's email messages which are stored on a mail server. However, unlike the Post Office Protocol (POP) where messages are downloaded and processed on the local system, the messages on an IMAP server are retained on the server and processed remotely. This is ideal for users who need access to a centralized store of messages or have limited bandwidth. For example, traveling salesmen who have notebook computers or mobile users on a wireless network would be ideal candidates for using IMAP. The SocketTools IMAP class implements the current standard for this protocol, and provides methods to retrieve messages, or just certain parts of a message, create and manage mailboxes, search for specific messages based on certain criteria and so on. The interface is designed as a superset of the Post Office Protocol interface, so developers who are used to working with the POP3 class will find the IMAP class very easy to integrate into an existing application.

SocketTools.InternetServer Class

The Internet Server ActiveX control provides a simplified interface for creating event-driven, multithreaded server applications using the TCP/IP protocol. The control interface is similar to the SocketWrench ActiveX control, however it is designed specifically to make it easier to implement a server application without requiring the need to manage multiple socket controls. In addition, the Internet Server control supports secure communications using the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols.

SocketTools.MailMessage Class

The Mail Message class provides an interface for composing and processing email messages and newsgroup articles which are structured according to the Multipurpose Internet Mail Extensions (MIME) standard. Using this class, an application can easily create complex messages which include multiple alternative content types, such as plain text and styled HTML text, file attachments and customized headers. It is not required that the developer understand the complex MIME standard; a single method call can be used to create multipart message, complete with a styled HTML text body and support for international character sets. The Mail Message class can be easily integrated with the other mail related protocol libraries, making it extremely easy to create and process MIME formatted messages.

SocketTools.NntpClient Class

The Network News Transfer Protocol (NNTP) is used with servers that provide news services. This is similar in functionality to bulletin boards or message boards, where topics are organized hierarchically into groups, called newsgroups. Users can browse and search for messages, called news articles, which have been posted by other users. On many servers, they can also post their own articles which can be read by others. The largest collection of public newsgroups available is called USENET, a world-wide distributed discussion system. In addition, there are a large number of smaller news servers. For example, Microsoft operates a news server which functions as a forum for technical questions and announcements. The class provides a comprehensive interface for accessing newsgroups, retrieving articles and posting new articles. In combination with the Mail Message class to process the news articles, SocketTools can be used to integrate newsgroup access with an existing email application, or you can implement your own full-featured newsgroup client.

SocketTools.NetworkTime Class

The NetworkTime class provides an interface for synchronizing the local system's time and date with that of a server. The class enables developers to query a server for the current time and then update the system clock if desired.

SocketTools.PopClient Class

The Post Office Protocol (POP) provides access to a user's new email messages on a mail server. Methods are provided for listing available messages and then retrieving those messages, storing them either in files or in memory. Once a user's messages have been downloaded to the local system, they are typically removed from the server. This is the most popular email protocol used by Internet Service Providers (ISPs) and the class provides a complete interface for managing a user's mailbox. This class is typically used in conjunction with the Mail Message class, which is used to process the messages that are retrieved from the server.

SocketTools.InternetDialer Class

The Remote Access Services (RAS) class enables an application to connect to an Internet Service Provider (ISP) using a standard Dial-Up Networking connection. Using this class, the application can discover what dial-up devices are available, what dial-up networking entries, known as "connectoids", are available on the local system and allows the program to manage those connections. Existing connections can be monitored, new connections created and a single class can be used to manage multiple dial-up connections if the system has more than one modem. While Windows can be configured to simply autodial a service provider whenever a network connection is needed, this component gives your application complete control over the process of connecting to a service provider, monitoring that connection and then terminating that connection if needed.

SocketTools.RshClient Class

The Remote Command protocol is used to execute a command on a server and return the output of that command to the client. The class provides an interface to this protocol, enabling applications to remotely execute a command and process the output. This is most commonly used with UNIX based servers, although there are implementations of remote command servers for the Windows operating system. The class supports both the rcmd and rshell remote execution protocols and provides methods which can be used to search the data stream for specific sequences of characters. This makes it extremely easy to write Windows applications which serve as light-weight client interfaces to commands being executed on a UNIX server or another Windows system. The class can also be used to establish a remote terminal session using the rlogin protocol, which is similar to how the Telnet protocol methods.

SocketTools.SshClient Class

The Secure Shell (SSH) protocol is used to establish a secure connection with a server which provides a virtual terminal session for a user. Its functionality is similar to how character based consoles and serial terminals work, enabling a user to login to the server, execute commands and interact with applications running on the server. The SSH control provides an interface for establishing the connection and handling the standard I/O functions needed by the program. The control also provides methods that enable a program to easily scan the data stream for specific sequences of characters, making it very simple to write light-weight client interfaces to applications running on the server.

SocketTools.SmtpClient Class

The Simple Mail Transfer Protocol (SMTP) enables applications to deliver email messages to one or more recipients. The class provides an interface for addressing and delivering messages, and extended features such as user authentication and delivery status notification. Unlike Microsoft's Messaging API (MAPI) or Collaboration Data Objects (CDO), there is no requirement to have certain third-party email applications installed or specific types of servers installed on the local system. The class can be used to deliver mail through a wide variety of systems, from standard UNIX based mail servers to Windows systems running Exchange or Lotus Notes and Domino. Using this class, messages can be delivered directly to the recipient, or they can be routed through a relay server, such as an Internet Service Provider's mail system. The Mail Message class can be integrated with this class in order to provide an extremely simple, yet flexible interface for composing and delivering mail messages.

SocketTools.SocketWrench Class

The SocketWrench class provides a higher-level interface to the Windows Sockets API, designed to be suitable for programming languages other than C and C++. In addition, SocketWrench supports secure communications using the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols.

SocketTools.TelnetClient Class

The Telnet protocol is used to establish a connection with a server which provides a virtual terminal session for a user. Its functionality is similar to how character based consoles and serial terminals work, enabling a user to login to the server, execute commands and interact with applications running on the server. The class provides an interface for establishing the connection, negotiating certain options (such as whether characters will be echoed back to the client) and handling the standard I/O functions needed by the program. The class also provides methods that enable a program to easily scan the data stream for specific sequences of characters, making it very simple to write light-weight client interfaces to applications running on the server. This class can be combined with the Terminal Emulation control to provide complete terminal emulation services for a standard ANSI or DEC-VT220 terminal.

SocketTools.Terminal Class

The Terminal Emulation control provides a comprehensive interface for emulating an ANSI or DEC-VT220 character terminal, with full support for all standard escape and control sequences, color mapping and other advanced features. The class methods provide both a high level interface for parsing escape sequences and updating a display, as well as lower level primitives for directly managing the virtual display, such as controlling the individual display cells, moving the cursor position and specifying display attributes. This control can be used in conjunction with the Remote Command or Telnet Protocol class to provide terminal emulation services for an application, or it can be used independently. For example, this control could also be used to provide emulation services for a program that provides serial modem connections to a server.

SocketTools.WebLocation Class

The WebLocation class provides your application with geographical information about the physical location of the computer system based on its external IP address. This can enable developers to know where their application is being used, and provide convenience functionality such as automatically completing a form based on the location of the user.

SocketTools.WebStorage Class

The WebStorage class provides private cloud storage for uploading and downloading shared data files which are available to your application. This is primarily intended for use by developers to store configuration information and other data generated by the application. For example, you may want to store certain application settings, and the next time a user or organization installs your software, those settings can be downloaded and restore.

SocketTools.WhoisClient Class

The Whois protocol provides a mechanism for requesting information about an Internet domain name. When a domain name is registered, the organization that registers the domain must provide certain contact information along with technical information such as the primary name servers for that domain. The Whois protocol enables an application to query a server which provides that registration information. The class provides an interface for requesting that information and returning it to the program so that it can be displayed or processed.

Domain Name Services

The Domain Name Service (DNS) protocol is what applications use to resolve domain names into Internet addresses, as well as provide other information about a domain, such as the name of the mail servers which are responsible for receiving email for users in that domain. All of the SocketTools components provide basic domain name resolution functionality, but the Domain Name Services class gives an application direct control over what servers are queried, the amount of time spent waiting for a response and the type of information that is returned.

Initialization

Initialize

Initialize an instance of the class, loading the networking library and validating the development license. This method must be called before any properties are changed or any other methods in this class are called by the application.

Reset

Reset the internal state of the class and re-initialize the component to use the default nameserver configuration for the local host. This can be useful if your application wishes to discard any settings made by a user and return to using the local system configuration.

Uninitialize

Unload the networking library and release any resources that have been allocated for the class instance. This method is automatically invoked when the class instance is disposed or goes out of scope.

Host Tables

When resolving a host name or IP address, the class will first search the local system's host table, a file that is used to map host names to addresses. The hosts file is found in C:\Windows\system32\drivers\etc\hosts. Note that the file does not have an extension.

HostFile

Return the full path of the file that contains the default host table for the local system. This can be useful if you wish to temporarily switch between the default host file and another host file specific to your application.

Host Name Resolution

The class can be used to resolve host names into IP addresses, as well as perform reverse DNS lookups converting IP addresses into the host names that are assigned to them. The class will search the local system's host table first, and then perform a nameserver query if required.

HostAddress

A property which returns the IP address of the host name specified in the HostName property. Setting this property to an IP address will cause the class to perform a reverse DNS lookup to attempt to determine the name of the host that was assigned that address. If successful, the host name for the specified IP address can be determined by reading the value of the HostName property.

HostName

A property which returns the name of the host associated with the IP address specified in the HostAddress property. Setting this property to a host name will cause the class to perform a DNS lookup to determine the IP address of that host. If successful, the IP address for the host can be determined by reading the value of the HostAddress property.

Resolve

A method which resolves a host name into an IP address, returned as a string in dotted notation. The class first checks the system's local host table, and if the name is not found there, it will perform a nameserver query for the A (address) record for that host.

Query
Perform a general nameserver query for a specific record type. This method can be used to perform queries for the common record types such as A and PTR records, as well as for other record types such as TXT (text) records. Refer to the Technical Reference for more information about the specific types of records that can be returned.

Mail Exchange Records

When a system needs to deliver a mail message to someone, it needs to determine what server is responsible for accepting mail for that user. This is done by looking up the mail exchange (MX) record for the domain. For example, if a message was addressed to joe@example.com, to determine the name of the mail server that would accept mail for that recipient, you would perform an MX record query against the domain example.com. A domain may have more than one mail server, in which case multiple MX records will be returned.

MailExchange
A property array which returns the mail exchanges for the domain specified in the HostName property. This is a zero-based array, with the maximum number of entries returned by the MailExchanges property

Advanced Properties

In addition to providing host name and IP address resolution, the class can be used to perform advanced queries for other types of records.

HostInfo
Return additional information about the specified host name. If the name server has been configured to provide host information for the domain, this method will return that data. Typically it is used to indicate what hardware and operating system the host uses.

HostServices
Return information about the UDP and TCP based services that the host provides. If defined, this will return a list of service names such as "ftp" and "http". Note that your application should not depend on this information to be a definitive list of what services a server provides.

NameServer
A property array which can be used to return the current nameservers that are configured for the local host, or the values can be changed to specify new nameservers. The maximum number of nameservers that can be configured for each instance of the class is four.

File Encoding

A common requirement for applications which use Internet protocols is the need to encode binary files, as well as compress data to reduce the bandwidth and time required to send or receive the data. Encoding a binary file converts the contents of the file into printable characters which can be safely transferred over the Internet using protocols that only support a subset of 7-bit ASCII characters. This is commonly a restriction for email, since many mail servers still are not capable of correctly processing messages which contain control characters, 8-bit data or multi-byte character sequences found in International text. To address this problem, the sender encodes and sends the data as part of a message; the recipient then extracts and decodes the data, with the end result being the same as the original, without any potential corruption by the mail servers which store and/or forward the message. The File Encoding class supports several encoding and decoding methods, including standard base64 encoding, quoted-printable encoding and uuencoding. For applications which access USENET newsgroup, the class also supports the newer yEnc encoding method which has become a popular method for attaching binary files to a message.

In addition to encoding and decoding files, the File Encoding class also can be used to compress files, reducing their overall size. Two compression algorithms are supported, the standard deflate algorithm which is commonly used in Zip files, and an algorithm based on the Burrows-Wheeler Transform (BWT) which can offer improved compression over the deflate algorithm for some types of files. The developer has control over the type of compression performed, as well as details such as the level of compression which determines how much memory and CPU time is allocated to compress the data. Developers can even create their own custom compression formats by creating an application-specific header block, typically represented by a structure or user-defined type that can be used to provide information to the program.

Note that if you are interested in using this class for purposes of attaching files to an email message, it is not necessary that you use these methods. The Mail Message class has the ability to automatically encode and decode file attachments without requiring that you use the methods in this control. However, the File Encoding class is useful if you need the ability to encode and/or compress for other applications.

Encoding Types

There are several different encoding types available, with the default being the standard MIME encoding called Base64. The following encoding methods are supported by the class:

Base64
Base64 encoding works by representing three bytes of data as four printable characters. Each of the three bytes is converted into four six-bit numbers, and each six-bit number is converted to one of 64 printable characters (which is where the encoding method gets its name). Base64 is the default encoding method used by the control and is the standard encoding used for MIME formatted email messages as well as many other applications.

Quoted-Printable
Quoted-printable encoding is primarily used in email messages, and is best used when the data being encoded is text which consists primarily of printable characters. Only characters with the high-bit set or a certain subset of printable characters are actually encoded by representing them as their hexadecimal value. All other printable characters are passed through unmodified.

Uucode
One of the original encoding methods used for email, it gets its name from two UNIX command-line utilities called uuencode and uudecode, which were used to encode and decode files. Like Base64, uuencoding converts three bytes of data into four six-bit numbers, and then a value of 32 is added to ensure that it is printable. Uuencoding also adds some additional characters which are used to ensure the integrity of the encoded data. This encoding method is still used when posting files to USENET

newsgroups, but has largely been replaced by Base64 when attaching files to email messages.

yEnc
yEnc is a relatively new encoding method that was created specifically for binary newsgroups on USENET. Because USENET doesn't have the same limitations as email systems in terms of what kind of characters can be safely used, yEnc only encodes null characters and certain control characters; the remaining 8-bit data is passed through as is which can significantly reduce the overall size of the encoded data. yEnc also uses checksums to ensure the integrity of the data and is designed so that a large file can be split across multiple messages and then recreated.

Data Encoding

Encoding a binary file converts the contents of the file into printable characters which can be safely transferred over the Internet using protocols that only support a subset of 7-bit ASCII characters. This is commonly a restriction for email, since many mail servers still are not capable of correctly processing messages which contain control characters, 8-bit data or multi-byte character sequences found in International text. To address this problem, the sender encodes and sends the data as part of a message; the recipient then extracts and decodes the data, with the end result being the same as the original, without any potential corruption by the mail servers which store and/or forward the message.

EncodeFile
This method encodes a file using the specified encoding method, storing the encoded data in a new file. An option also allows you to automatically compress the data prior to encoding it in order to reduce the overall size of the encoded file.

DecodeFile
This method decodes a previously encoded file using the specified encoding method, restoring the original contents. If the encoded data was compressed, this method can also be used to automatically expand the data after it has been decoded.

Data Compression

In addition to encoding and decoding data, the control can be used to compress data in order to reduce its size. The compression methods may be used separately, or may be used as part of the process of encoding a file.

CompressFile
This method reduces the size of a file using the standard Deflate algorithm. This is the same algorithm that is commonly used in Zip archives. Note however, that this does not create a Zip file, it simply uses the same compression method.

ExpandFile
This method restores the original contents of a file that was previously compressed using the CompressFile method. Note that this method is not designed to extract files from a Zip archive or expand data compressed using a different algorithm.

Note that there are advanced options for compressing files, such as the ability to specify the compression type and level. Please refer to the Technical Reference for more information.

File Transfer Protocol

The File Transfer Protocol (FTP) is the most common application protocol used to upload and download files. In addition to basic file transfer capabilities, FTP also supports common file and directory management functions on the server, such as renaming and deleting files or creating new directories. The SocketTools .NET Edition also supports secure file transfers using SSH (SFTP) and SSL/TLS (FTPS) by simply specifying an option when establishing the connection.

Initialization

Initialize
Initialize an instance of the class, loading the networking library and validating the development license. This method must be called before any properties are changed or any other methods in this class are called by the application.

Connect
Connect to the server, using either a host name or IP address. The method has several options related to security as well as the general operation of the class. One important option is **FtpOptions.optionPassive**, which instructs the class to use passive mode file transfers. If the local system is behind a firewall or a route which uses Network Address Translation (NAT), it is often necessary to use this option. It is also possible to enable passive mode transfers by simply setting the Passive property prior to calling this method.

Login
Authenticate the client session, providing the server with a user name, password and optionally an account name. It is also possible to use an anonymous (unauthenticated) session by providing empty strings as the username and password. If the **UserName** and **Password** properties are set prior to connecting, the user will automatically be logged in. This method is only necessary if the application needs to access the server using different user accounts during the same session.

Disconnect
Disconnect from the server and release the memory allocated for that client session. After this method is called, the client session is no longer valid.

Reset
Reset the internal state of the component. This can be useful if your application wishes to discard any settings made by a user and return that instance of the class to its default state.

Uninitialize
Unload the networking library and release any resources that have been allocated for the class instance. This method is automatically invoked when the class instance is disposed or goes out of scope.

File Transfers

The class provides several methods which can be used to transfer files between the local and server. This group of methods are high level, meaning that it is not necessary to actually write the code to read and/or write the file data. The class automatically handles the lower level file I/O and notifies your application of the status of the transfer by periodically generating progress events.

GetData
This method transfers a file from the server to the local system, storing the file data in memory. This can be useful if your application needs to perform some operation based on the contents of the file, but does not need to store the file locally. The file data can returned in a string or byte array.

GetFile
This method transfers a file from the server and stores it in a file on the local system. This method is similar to how the GET command works for the command-line FTP client in Windows.

GetMultipleFiles

This method transfers multiple files from the server and stores them in a directory on the local system. A wildcard may be specified so that only files which a certain name or those that match a particular file extension are downloaded. This method is similar to how the MGET command works for the command-line FTP client in Windows.

PutData
This method creates a file on the server containing the data that you provide. This can be useful if your application wants to upload dynamically created content without having to create a temporary file on the local system. The data may be specified either as a string, or as the contents of a byte array.

PutFile
This method uploads a file from the local system to the server. This method is similar to how the PUT command works for the command-line FTP client in Windows.

PutMultipleFiles
This method transfers multiple files from the local system to a directory on the server. A wildcard may be specified so that only files with a certain name or those that match a particular file extension are uploaded. This method is similar to how the MPUT command works for the command-line FTP client in Windows.

File Management

In addition to performing file transfers, the File Transfer Protocol class can also perform many of the same kinds of file management methods on the server as you would on the local system.

DeleteFile
Delete a file from the server. This operation requires that the current user have the appropriate permissions to delete the file.

GetFileStatus
Return status information about the file in the form of a structure. This typically specifies the ownership, access permissions, size and modification time for the file. It is similar to opening a directory on the server and reading information about the file, but with less overhead.

GetFileSize
Return the size of a file on the server without actually downloading the contents of the file.

GetFileTime
Return the modification time for the specified file on the server. This can be used by you application to determine if the file has been changed since the time that you last uploaded or downloaded the contents.

RenameFile
Change the name of a file or move a file to a different directory. This operation requires that the current user have the appropriate permissions to rename the file. If the file is being moved to another directory, the user must have permission to access that directory.

SetFileTime
Update the modification time for a file on the server. This method requires that the current user have the appropriate permissions to change the last modification timestamp for the file. Note that this is not supported on all servers and in some cases may be restricted to specific accounts.

GetFilePermissions
Return the access permissions for a file on the server. This can be used to determine if a file can be read, modified and/or deleted by the current user. For users who are familiar with UNIX file permissions, it is the same type which is used by the class.

SetFilePermissions
Change the access permissions for a file. This method is supported on most UNIX based servers, as well as any other server that supports the site-specific CHMOD command.

Directory Management

The class also provides a set of methods which can be used to access and manage directories or folders, including the ability to list and search for files, create new directories and remove empty directories from the server.

ChangeDirectory
Change the current working directory on the server. This is similar to how the CD command is used from the command-line to change the current directory in Windows. If a path is not specified in the file name, the current working directory is where files will be uploaded to and downloaded from.

MakeDirectory
Create a new directory on the server. This requires that the current user have the appropriate access permissions in order to create the directory.

OpenDirectory
Open the specified directory on the server. This is the first step in returning a list of files in the directory. After the directory has been opened, information about the files it contains can be returned to the application. The directory path may also include wildcards to only return information about a certain subset of files based on the file name or extension.

GetFirstFile and GetNextFile
Return information about the next file in the directory that has been opened. This method is called repeatedly until it indicates that all of the files have been returned.

RemoveDirectory
Remove an empty directory from the server. This operation requires that the current user have the appropriate permissions to delete the directory. For safety, it is required that the directory does not contain any files or subdirectories or the operation will fail.

File Transfer Server

The File Transfer Server class provides an interface for implementing an embedded, lightweight server that can be used to exchange files with a client using the standard File Transfer Protocol. The server can accept connections from any third-party application or a program developed using SocketTools. The application specifies an initial server configuration by setting the relevant properties and can implement event handlers to monitor the activities of the clients that have connected to the server. The class automatically handles the standard FTP commands and requires minimal coding on the part of the application that is hosting the control. However, the application may also use event mechanism to filter specific commands or to extend the protocol by providing custom implementations of existing commands or add entirely new commands.

An important consideration when using this class is that events are raised in the context of the thread that manages the client session. The .NET Framework does not allow one thread to modify a control that was created in the main user interface thread, which means that you cannot update user interface controls directly from within the event handlers. If you want to change any property values or call methods in a control, you need to create a delegate and marshal the call to the user interface thread to using the control's Invoke method.

Initialization

Initialize
Initialize an instance of the class, loading the networking library and validating the development license. This method must be called before any properties are changed or any other methods in this class are called by the application.

Start
This method starts the server, creating the background thread and listening for incoming client connections on the specified port number. You can specify the local address, port number, backlog queue size and the maximum number of clients that can establish a connection with the server.

Restart
This method will terminate all active client connections, close the listening socket and re-create a new listening socket bound to the same address and port number.

Suspend
This method instructs the server to temporarily suspend accepting new client connections. Existing connections are unaffected, and any incoming client connections are rejected until the server is resumed. It is not recommended that you leave a server in a suspended state for an extended period of time.

Resume
This function instructs the server to resume accepting client connections after it was suspended. Any pending client connections are accepted after the server has resumed normal operation.

Throttle
This method is used to control the maximum number of clients that may connect to the server, the maximum number of clients that can connect from a single IP address and the rate at which the server will accept client connections. By default, there are no limits on the number of active client sessions and connections are accepted immediately. This method can be useful in preventing denial-of-service attacks where the the attacker attempts to flood the server with connection attempts.

Stop
This method will terminate all active client connections, close the listening socket and terminate the background thread that manages the server. Any incoming client connections will be refused, and all resources allocated for the server will be released.

Uninitialize

Unload the networking library and release any resources that have been allocated for the class instance. This method is automatically invoked when the class instance is disposed or goes out of scope.

Events

The application is informed of all client activity through event notifications. These events will tell your program when a client has connected, issued a command, uploaded or downloaded a file or disconnects from the server. Through the event mechanism it's also possible to implement your own custom commands.

OnConnect

This event occurs when the client first establishes a connection with the server. At this point, the client is in an unauthenticated state and may only issue a limited subset of commands before it submits user credentials for authentication. This event provides your application with the unique client ID that identifies the session and the remote IP address that the client has connected from.

OnAuthenticate

This event occurs when the client has submitted user credentials for authentication. The class supports several mechanisms for user authentication, including automatic local user authentication, the creation of one or more virtual users or custom authentication by implementing an event handler for this event. It is not necessary to implement a handler for this event if you choose to create virtual user accounts for your server instance because the server will automatically handle authentication for those users.

OnCommand

This event occurs for each command sent by the client, prior to the command being processed by the server. This event can be used by your application to monitor the commands that are being issued, or may create an event handler that filters certain commands or implements support for custom commands by processing the command in your own code.

OnResult

This event occurs after each command has been processed by the server. This event can be used by your application to monitor those commands which were successful and those which failed. For example, your application could use this event to track the actions of a client and terminate the session if the client is issuing a large number of commands that fail.

OnDownload

This notification event occurs after a file has been successfully downloaded by a client. It can be used for logging and monitoring purposes or to update your user interface with relevant information about the current client activity.

OnUpload

This notification event occurs after a file has been successfully uploaded by a client.

OnDisconnect

This event occurs after a client has disconnected from the server and the thread that manages the client session has released all of the resources allocated for that client.

Local Host Information

Several properties are provided to return information about the local host, including its fully qualified domain name and the IP addresses that are configured on the system.

ServerName

Return the fully qualified domain name of the local host, if it has been configured. If the system has not been configured with a domain name, then the machine name is returned instead.

ExternalAddress

Return the IP address assigned to the router that connects the local host to the Internet. This is typically used by an application executing on a system in a local network that uses a router which performs

Network Address Translation (NAT).

AdapterAddress
This property array returns the IP addresses that are associated with the local network or remote dial-up network adapters configured on the system. The AdapterCount property can be used to determine the number of adapters that are available.

Hypertext Transfer Protocol

The Hypertext Transfer Protocol (HTTP) is the most prevalent application protocol used on the Internet today. It was originally used for document retrieval, and has grown into a complex protocol which supports file uploading, script execution, file management and distributed web authoring through extensions such as WebDAV. The SocketTools Hypertext Transfer Protocol control implements version 0.9, 1.0 and 1.1 of the protocol, including features such as support for proxy servers, persistent connections, user-defined header fields and chunked data.

File Transfers

Similar to the interface used with the File Transfer Protocol control, you can use HTTP to upload and download files. In addition to the standard method for downloading files, the control supports two methods for uploading files, using either the PUT or the POST command. When downloading a file from the server, you can either store the contents in a local file, or you can copy the data into a memory buffer that you allocate. Similarly, when uploading files, you can either specify a local file to upload, or you can provide a memory buffer that contains the file data to send to the server. High level methods such as PutFile and GetFile can be used to transfer files in a single method call. There are also methods such as OpenFile and CreateFile which provide lower level file I/O interfaces.

Script Execution

Another common use for HTTP is to execute scripts on the web server. The application can pass additional data to the script, which is similar in concept to how arguments are passed to a command that is entered from the command prompt. This uses the standard POST command, and the resulting output from the script is returned back to the application where it can be displayed or processed. An application can use the Command method to execute the script and then process the output in code, or can use the higher level method PostData which will execute the script and return the output from that script in a single method call.

Uniform Resource Locators

Anyone who has used a web browser is familiar with the Uniform Resource Locator (URL); it is the value that is entered as the address of a website. URLs have a specific format which provides information about the server, the port number and the name of the resource that is being accessed:

```
http://[username : [password] @] remotehost [:remoteport] / resource [?
parameters]
```

The first part of the URL identifies the protocol, also known as the scheme, which will be used. With web servers, this will be either http or https for secure connections. If a username and password is required for authentication, then this will be included in the URL before the name of the server. Next, there is the name of the server to connect to, optionally followed by a port number. If no port number is given, then the default port for the protocol will be used. This is followed by the resource, which is usually a path to a file or script on the server. Parameters to the resource may also be specified, called the query string, which are typically used as arguments to a script that is executed on the server.

Understanding how a URL is constructed will help in understanding how the different methods in the control work together. For example, the server name and port number portion of the URL are the values passed to the Connect method to establish the connection. The user name and password values are assigned to the UserName and Password properties to authenticate the client session. And the resource name is passed to the GetData or GetFile methods to transfer it to the local system.

Initialization

Initialize

Initialize an instance of the class, loading the networking library and validating the development license. This method must be called before any properties are changed or any other methods in this class are called by the application.

Connect
Connect to the server, using either a host name or IP address. This method creates the client session and must be called before your application attempts to request a resource from the server.

Disconnect
Disconnect from the server and release the memory allocated for that client session. After this method is called, the client session is no longer valid.

Reset
Reset the internal state of the component. This can be useful if your application wishes to discard any settings made by a user and return that instance of the class to its default state.

Uninitialize
Unload the networking library and release any resources that have been allocated for the class instance. This method is automatically invoked when the class instance is disposed or goes out of scope.

File Transfers

Using an interface similar to the File Transfer Protocol control, this control provides several methods which can be used to transfer files between the local and server. This group of methods is high level, meaning that it is not necessary to actually write the code to read and/or write the file data. The control automatically handles the lower level file I/O and notifies your application of the status of the transfer by periodically generating progress events.

GetData
This method transfers a file from the server to the local system, storing the file data in memory. This can be useful if your application needs to perform some operation based on the contents of the file, but does not need to store the file locally.

GetFile
This method transfers a file from the server and stores it in a file on the local system.

PutData
This method creates a file on the server containing the data that you provide. This can be useful if your application wants to upload dynamically created content without having to create a temporary file on the local system.

PutFile
This method uploads a file from the local system to the server using the PUT command. Not all servers support this command, and some may require that the client authenticate prior to calling this method.

PostFile
This method uploads a file from the local system to the server using the POST command. This enables your application to upload a file in the same way that a user would when using a form in a web browser.

File Management

The control can also perform some basic file management methods as well as send custom commands to the server. Some web servers also provide more advanced document management methods using WebDAV, an extension to HTTP for distributed document authoring.

GetFileSize
Return the size of a file on the server without actually downloading the contents of the file. It is important to note that most servers will only return file size information for actual documents stored on the server, not for dynamically created content generated by scripts or web pages which use server-side includes.

GetFileTime
Return the modification time for the specified file on the server. This can be used by your application to determine if the file has been changed since the time that you last uploaded or downloaded the contents.

DeleteFile

Remove a file from the server. This operation requires that the current user have the appropriate permissions to delete the file. Not all servers support the use of this command, and it would typically require that the client authenticate prior to calling this method.

Command

This method enables the client to send any command directly to the server. This is commonly used to issue custom commands to servers that are configured to use extensions to the standard protocol.

Script Execution

The control also provides methods to execute scripts on the web server and return the output from those scripts back to your application. Your program can pass additional data to the script, typically either as a query string or as form data, which is similar in concept to how arguments are passed to a command that is entered from the command prompt.

GetData

In addition to being used to simply return the contents of a file, this method can also be used to execute a script on the server and return the output of that script to your program. Arguments to the script can be specified by passing them as a query string. For example, consider the following resource name:

```
/cgi-bin/test.cgi?data1=value1&data2=value2
```

This would specify that the script /cgi-bin/test.cgi will be executed, and two arguments will be passed to that script: data1=value and data2=value2. The ampersand is used to separate the arguments, and they are grouped as pairs of values separated by an equal sign. Note that the actual format and value of the query string depends on how the script is written.

PostData

An alternative method of providing information to a script is to post data to the script. Instead of the data being part of the resource name itself, posted data is sent separately and is provided as input to the script. This is the same method that is typically used when a user clicks the Submit button on a web-based form. This method requires the name of the script and the address of a buffer that contains the data that will be posted. The resulting output from the script is returned to the caller in the same way that the GetData method works.

Hypertext Transfer Server

The Hypertext Transfer Server class provides an interface for implementing an embedded, lightweight server that can be used to provide access to documents and other resources using the Hypertext Transfer Protocol. The application specifies an initial server configuration and then responds to events that are generated when the client sends a request to the server. An application may implement only minimal handlers for most events, in which case the default actions are performed for most standard HTTP commands. However, an application may also use the event mechanism to filter specific commands or to extend the protocol by providing custom implementations of existing commands or add entirely new commands.

An important consideration when using this class is that events are raised in the context of the thread that manages the client session. The .NET Framework does not allow one thread to modify a control that was created in the main user interface thread, which means that you cannot update user interface controls directly from within the event handlers. If you want to change any property values or call methods in a control, you need to create a delegate and marshal the call to the user interface thread to using the control's Invoke method.

Initialization

Initialize

Initialize an instance of the class, loading the networking library and validating the development license. This method must be called before any properties are changed or any other methods in this class are called by the application.

Start

This method starts the server, creating the background thread and listening for incoming client connections on the specified port number. You can specify the local address, port number, backlog queue size and the maximum number of clients that can establish a connection with the server.

Restart

This method will terminate all active client connections, close the listening socket and re-create a new listening socket bound to the same address and port number.

Suspend

This method instructs the server to temporarily suspend accepting new client connections. Existing connections are unaffected, and any incoming client connections are rejected until the server is resumed. It is not recommended that you leave a server in a suspended state for an extended period of time.

Resume

This function instructs the server to resume accepting client connections after it was suspended. Any pending client connections are accepted after the server has resumed normal operation.

Throttle

This method is used to control the maximum number of clients that may connect to the server, the maximum number of clients that can connect from a single IP address and the rate at which the server will accept client connections. By default, there are no limits on the number of active client sessions and connections are accepted immediately. This method can be useful in preventing denial-of-service attacks where the the attacker attempts to flood the server with connection attempts.

Stop

This method will terminate all active client connections, close the listening socket and terminate the background thread that manages the server. Any incoming client connections will be refused, and all resources allocated for the server will be released.

Uninitialize

Unload the networking library and release any resources that have been allocated for the class instance.

This method is automatically invoked when the class instance is disposed or goes out of scope.

Events

The application is informed of all client activity through event notifications. These events will tell your program when a client has connected, issued a command, uploaded or downloaded a file or disconnects from the server. Through the event mechanism it's also possible to implement your own custom commands.

OnConnect
This event occurs when the client first establishes a connection with the server. This event provides your application with the unique client ID that identifies the session and the remote IP address that the client has connected from.

OnAuthenticate
This event occurs when the client has submitted user credentials for authentication. The class supports several mechanisms for user authentication, including automatic local user authentication, the creation of one or more virtual users or custom authentication by implementing an event handler for this event. It is not necessary to implement a handler for this event if you choose to create virtual user accounts for your server instance because the server will automatically handle authentication for those users.

OnCommand
This event occurs for each command sent by the client, prior to the command being processed by the server. This event can be used by your application to monitor the commands that are being issued, or may create an event handler that filters certain commands or implements support for custom commands by processing the command in your own code.

OnResult
This event occurs after each command has been processed by the server. This event can be used by your application to monitor those commands which were successful and those which failed. For example, your application could use this event to track the actions of a client and terminate the session if the client is issuing a large number of commands that fail.

OnDownload
This notification event occurs after a file has been successfully downloaded by a client. It can be used for logging and monitoring purposes or to update your user interface with relevant information about the current client activity.

OnUpload
This notification event occurs after a file has been successfully uploaded by a client.

OnDisconnect
This event occurs after a client has disconnected from the server and the thread that manages the client session has released all of the resources allocated for that client.

Local Host Information

Several properties are provided to return information about the local host, including its fully qualified domain name and the IP addresses that are configured on the system.

ServerName
Return the fully qualified domain name of the local host, if it has been configured. If the system has not been configured with a domain name, then the machine name is returned instead.

ExternalAddress
Return the IP address assigned to the router that connects the local host to the Internet. This is typically used by an application executing on a system in a local network that uses a router which performs Network Address Translation (NAT).

AdapterAddress

This property array returns the IP addresses that are associated with the local network or remote dial-up network adapters configured on the system. The AdapterCount property can be used to determine the number of adapters that are available.

Internet Control Message Protocol

The Internet Control Message Protocol (ICMP) class enables your application to send and receive ICMP echo datagrams. These are a special type of IP datagram which can be used to determine if a server is reachable, as well as determine the amount of time it takes for data to be exchanged with the local system. The ICMP class can also be used to trace the route that data takes from the local system to a server, which can be useful in determining why a connection to a particular system may be experiencing higher latency than normal.

Overview

Initialize
Initialize an instance of the class, loading the networking library and validating the development license. This method must be called before any properties are changed or any other methods in this class are called by the application.

Reset
Reset the internal state of the component. This can be useful if your application wishes to discard any settings made by a user and return that instance of the class to its default state.

Uninitialize
Unload the networking library and release any resources that have been allocated for the class instance. This method is automatically invoked when the class instance is disposed or goes out of scope.

Ping and TraceRoute

To determine if a server is reachable, your application can send ICMP echo datagrams. You can also map the route between the local system and the server by sending a series of echo datagrams to each intermediate host. This is what the ping.exe and tracert.exe command line utilities do, and you can emulate that functionality in your own applications.

Echo
This is the simplest method you can use to send ICMP echo datagrams. Specify the server, the size of the ICMP datagram you want to send and the number of times you want to send it. The method will return if the operation was successful along with information such as the average number of milliseconds it took for the datagram to be returned by the server.

TraceRoute
This method will map the route that data packets take from your local system to a server. Whenever you send data over the Internet, that data is routed from one computer system to another until it reaches its destination. This method returns statistical information about each system that the data is routed through, and the latency between that system and the local host. For each intermediate host in the route to the destination server, the OnTrace event will fire.

OnTrace
This event is generated when the TraceRoute method is called. The event will fire for each intermediate host in the route from the local system and the server.

Remote Access Services

The Remote Access Services (RAS) InternetDialer class enables an application to connect to an Internet Service Provider (ISP) using a standard Dial-Up Networking connection. Using this class, the application can discover what dial-up devices are available, what dial-up networking entries, known as "connectoids", are available on the local system and allows the program to manage those connections. Existing connections can be monitored, new connections created and a single instance of the class can be used to manage multiple dial-up connections if the system has more than one modem. While Windows can be configured to simply autodial a service provider whenever a network connection is needed, this component gives your application complete control over the process of connecting to a service provider, monitoring that connection and then terminating that connection if needed.

Initialization

Initialize

Initialize an instance of the class, validating the development license. This method must be called before any properties are changed or any other methods in this class are called by the application.

Connect

Establish a connection to the dial-up networking server. Once the connection has been established, the class will authenticate the session and the local system will have a network connection to the service provider.

Disconnect

Disconnect from the server and release any resources that have been allocated for the dial-up networking session. After this method is called, the session is no longer valid.

Uninitialize

Release any resources that have been allocated for the class instance. This method is automatically invoked when the class instance is disposed or goes out of scope.

Connection Properties

The class properties are used to set or return information about the current dial-up networking connection. To load a dial-up networking connection, called a connectoid or phonebook entry, use the LoadEntry method. There are a large number of properties, however the most significant of those properties are as follows:

DeviceName

This property specifies the name of the device that is used to establish the dial-up networking connection. In most cases this is the name of an analog modem using a serial communications port, connected to a standard telephone line. If your application needs to enumerate the available dial-up networking devices, refer to the DeviceCount, DeviceEntry and DeviceType properties.

DynamicAddress

This property determines if the dial-up networking connection uses a dynamically assigned IP address returned by the server, or a specific IP address configured on the local host. In most cases, this property should be set to True, unless otherwise specified by your service provider.

DynamicNameservers

This property determines if the dial-up networking connection uses dynamically assigned nameservers, used to resolve domain names into IP addresses. In most cases, this property should be set to True. If your service provider requires that you explicitly specify the nameservers to use, then set this property to False and set the NameServer property array to the address of the nameserver(s) to use.

EntryName

This property specifies the name of the connectoid for the current dial-up networking connection. If no connection is active and no connectoid has been loaded, then this property will return an empty string.

InternetAddress
This property returns the IP address assigned to the current dial-up networking session, if a connection has been established. It can also be used to explicitly specify an IP address if the DynamicAddress property is set to False.

NameServer
This is a property array which specifies the IP addresses of the nameservers that are to be used for the current dial-up networking session. If a connection has been established, this property array will return the addresses of those nameservers that have been assigned to you. If the DynamicNameserver property is set to False, this property array can also be used to explicitly specify the nameservers to be used by the dial-up networking connection.

Password
This property specifies the password used to authenticate the dial-up networking connection.

PhoneNumber
This property specifies the telephone number for the dial-up networking connection. You should also check the value of the CountryCode property, which will tell your application if area code dialing rules are being used. If the CountryCode property is set to zero, then no area code dialing rules are in effect and the telephone number is dialed as-is. Otherwise you should check the value of the AreaCode property if you need to determine the area code being used for the connection.

UserName
This property specifies the username used to authenticate the dial-up networking connection

Managing Connectoids

A connectoid contains the information needed to establish a connection, and is represented as the icon in the Network Connections for the local system. Connectoids are referenced by name and typically are named after the service provider, such as "EarthLink" or "Verizon". In addition to simply connecting to a dial-up networking server, the class also enables your application to create, edit and delete these connectoids. Note that in the class documentation, connectoids are also referred to as "entry names" or "phonebook entries".

CreateEntry
This method displays a dialog box that allows the user to specify the information needed to create a new connectoid. This is similar to the dialog that is displayed whenever the user chooses to create a new Dial-Up Networking connection. Note that if you want to create a connectoid without showing a dialog to the user, use the SaveEntry method instead.

DeleteEntry
This method deletes an existing dial-up networking connection. Exercise caution when using this method; once a connectoid has been deleted, there is no way to recover it.

LoadEntry
This method loads an existing connectoid, and updates the properties to reflect the connectoid's settings. Changing one or more of those properties and then calling the SaveEntry method is how you can modify an existing connectoid.

RenameEntry
This method renames an existing connectoid.

SaveEntry
This method modifies or creates a new connectoid based on the current properties of the class instance. If the connectoid already exists, it is modified, otherwise a new connectoid is created. Unlike the CreateEntry method, this method will not display any dialogs, so it is the responsibility of the application to provide a user interface if needed.

Internet Message Access Protocol

The Internet Message Access Protocol (IMAP) is an application protocol which is used to access a user's email messages which are stored on a mail server. However, unlike the Post Office Protocol (POP) where messages are downloaded and processed on the local system, the messages on an IMAP server are retained on the server and processed remotely. This is ideal for users who need access to a centralized store of messages or have limited bandwidth. The SocketTools IMAP class implements the current standard for this protocol, and provides methods to retrieve messages, create and manage mailboxes, and search for specific messages based on some user-defined search criteria.

Initialization

Initialize
Initialize an instance of the class, loading the networking library and validating the development license. This method must be called before any properties are changed or any other methods in this class are called by the application.

Connect
Establish a connection to the IMAP server. Once the connection has been established, the other methods in the class may be used to access the messages on the server.

Disconnect
Disconnect from the server and release the memory allocated for that client session. After this method is called, the client session is no longer valid.

Reset
Reset the internal state of the component. This can be useful if your application wishes to discard any settings made by a user and return that instance of the class to its default state.

Uninitialize
Unload the networking library and release any resources that have been allocated for the class instance. This method is automatically invoked when the class instance is disposed or goes out of scope.

Managing Mailboxes

One of the primary differences between the IMAP and POP3 protocol is that IMAP is designed to manage messages on the mail server, rather than downloading all of the messages and storing them on the local system. To support this, IMAP allows the client to maintain multiple mailboxes on the server, which are similar in concept to message folders used by mail client software. A mailbox can contain messages, and in some cases a mailbox can contain other mailboxes, forming a hierarchy of mailboxes and messages, similar to directories and files in a filesystem. A special mailbox named INBOX contains new messages for the user, and additional mailboxes can be created, renamed and deleted as needed. The following are the most important methods for managing mailboxes:

CheckMailbox
Check the mailbox for any new messages which may have arrived. Because messages are managed on the server, it is possible for new mail to arrive during the client session.

CreateMailbox
Create a new mailbox on the server with the specified name.

DeleteMailbox
Delete a mailbox from the server. Most servers will only permit a mailbox to be deleted if it does not contain any mailboxes itself. Unlike deleting a message, which can be undeleted, deleting a mailbox is permanent.

ExamineMailbox
Once the session has been established and authenticated, a mailbox should be selected. This enables the

client to manage the messages in that mailbox. This method selects the specified mailbox in read-only mode so that messages can be read, but not modified. To select the mailbox in read-write mode, use the SelectMailbox method.

RenameMailbox
Renames an existing mailbox. One of the interesting uses of this method is the ability to rename the special INBOX mailbox. Instead of actually renaming it, it moves all of the messages to the new mailbox and empties the INBOX.

SelectMailbox
Once the session has been established and authenticated, a mailbox should be selected. Selecting a mailbox enables the client to manage the messages in that mailbox. This method selects the specified mailbox in read-write mode so that changes can be made to the mailbox.

UnselectMailbox
This method unselects the currently selected mailbox, and allows the caller to specify if messages marked for deletion should be expunged (removed) from the mailbox or reset back to an undeleted state.

Managing Messages

There are methods in the IMAP class for managing messages which enables the application to create, delete and move messages. To use these methods, a mailbox must be selected, either by setting the MailboxName property or calling the SelectMailbox method. Methods which modify the mailbox require that it be opened in read-write mode. Messages are identified by a number, starting with one for the first message in the mailbox.

CopyMessage
Copy a message to a specific mailbox.

DeleteMessage
Mark the specified message for deletion. Unlike the POP3 protocol, when a message is deleted on an IMAP server it can still be accessed. The message will not actually be removed from the mailbox unless the mailbox is expunged, unselected or the client disconnects from the server.

UndeleteMessage
Remove the deletion flag from the specified message.

Viewing Messages

One of the more powerful features of the IMAP protocol is the ability to precisely select what kinds of message data you wish to retrieve from the server. It is possible to retrieve only specific headers, or specific sections of a multipart message. Because IMAP understands MIME formatted messages, it is possible to only retrieve the textual portion of a message without having to download any attachments that may have come with it.

GetHeader
This method returns the value for a specified header field in the message. Using this method, it is not necessary to download and parse the message header.

GetHeaders
This method retrieves the complete headers for the specified message and stores it in a string or byte array provided by the caller.

GetMessage
The GetMessage method retrieves the specified message and stores it in a string or byte array provided by the caller; you can specify the type of message data that you want, a specific part of a multipart message and the amount of data that you want. For example, it is possible to request that only the first 1500 bytes of the body of the 3rd part of a multipart message should be returned.

OpenMessage

The OpenMessage method is a lower level method which opens a message for reading from the server. The application would then call Read to read the contents of the message, followed by CloseMessage when all the message data has been read. Also see the GetMessage method, which will return the contents of a message into a string or byte array.

Downloading Messages

In some cases, it may be preferable to download a complete message from the server to the local system. This can be easily done with a single method call.

StoreMessage
This method downloads a complete message and stores it as a text file on the local system.

Internet Server

The Internet Server class provides an interface which is similar to the SocketWrench control, but is specifically designed to simplify the development of a server application. The class provides a collection of methods which can be used to easily create an event-driven server application. The server runs on a separate thread in the background, automatically managing the individual client sessions as servers connect and disconnect from the server. Events are used to notify the application when the client establishes a connection with the server, sends data to the server or disconnects. Methods such as Read and Write are used to exchange data with the clients.

An important consideration when using the Internet Server class is that events are raised in the context of the thread that manages the client session. The .NET Framework does not allow one thread to modify a control that was created in the main user interface thread, which means that you cannot update user interface controls directly from within the event handlers. If you want to change any property values or call methods in a control, you need to create a delegate and marshal the call to the user interface thread to using the control's Invoke method.

Initialization

Initialize

Initialize an instance of the class, loading the networking library and validating the development license. This method must be called before any properties are changed or any other methods in this class are called by the application.

Start

This method starts the server, creating the background thread and listening for incoming client connections on the specified port number. You can specify the local address, port number, backlog queue size and the maximum number of clients that can establish a connection with the server.

Restart

This method will terminate all active client connections, close the listening socket and re-create a new listening socket bound to the same address and port number.

Suspend

This method instructs the server to temporarily suspend accepting new client connections. Existing connections are unaffected, and any incoming client connections are queued until the server is resumed. It is not recommended that you leave a server in a suspended state for an extended period of time. Once the connection backlog queue has filled, any subsequent client connections will be automatically rejected.

Resume

This function instructs the server to resume accepting client connections after it was suspended. Any pending client connections are accepted after the server has resumed normal operation.

Throttle

This method is used to control the maximum number of clients that may connect to the server, the maximum number of clients that can connect from a single IP address and the rate at which the server will accept client connections. By default, there are no limits on the number of active client sessions and connections are accepted immediately. This method can be useful in preventing denial-of-service attacks where the the attacker attempts to flood the server with connection attempts.

Lock

This method enables the application to lock the server so that only the current thread may interact with the server and the client sessions. This will cause all other client threads to go to sleep, waiting for the server to be unlocked. This should only be used when the server application needs to ensure that no other client threads are performing a network operation. If the server is left in a locked state for an extended period of time, it will cause the server to become non-responsive. If the application has started multiple servers, only one server can be locked at any one time.

Unlock
This method unlocks a server that has been previously locked. The threads which manage the client sessions will awaken and resume normal execution.

Stop
This method will terminate all active client connections, close the listening socket and terminate the background thread that manages the server. Any incoming client connections will be refused, and all resources allocated for the server will be released.

Uninitialize
Unload the networking library and release any resources that have been allocated for the class instance. This method is automatically invoked when the class instance is disposed or goes out of scope.

Input and Output

When a TCP connection is established, data is sent and received as a stream of bytes. The following methods can be used to send and receive data over the socket:

Read
This method reads data from the client and copy it to the string buffer or byte array provided by the caller. If the client closes its connection, this method will return zero after all the data has been read. If the method is successful, it will return the actual number of bytes read. This method should always be used when reading binary data from the client into a byte array.

ReadLine
Read a line of text from the client, up to an end-of-line character sequence or when the client closes the connection. This method is useful when the client and server are exchanging textual data, as is common with most command/response application protocols.

Write
This method sends data to the client. If the method succeeds, the return value is the number of bytes actually written. This method should always be used when sending binary data to the client.

WriteLine
Write a line of text to the socket, terminating it with an end-of-line character sequence. This method is useful when the client and server are exchanging textual data, as is common with most command/response application protocols.

Broadcast
Broadcasts data to each of the clients that are connected to the server. This can be useful when the application needs to send the same data to each active client session, such as broadcasting a shutdown message when the server is about to be terminated.

IsReadable
This property is used to determine if there is data available to be read from the socket. If the property returns a value of True, the Read method will return without causing the application to block. If the property returns False, there is no data available to read from the socket.

IsWritable
This property is used to determine if data can be written to the socket. In most cases this will return True, unless the internal socket buffers are full.

Local Host Information

Several properties are provided to return information about the local host, including its fully qualified domain name and the IP addresses that are configured on the system.

ServerName
Return the fully qualified domain name of the local host, if it has been configured. If the system has not

been configured with a domain name, then the machine name is returned instead.

ExternalAddress
Return the IP address assigned to the router that connects the local host to the Internet. This is typically used by an application executing on a system in a local network that uses a router which performs Network Address Translation (NAT).

AdapterAddress
This property array returns the IP addresses that are associated with the local network or remote dial-up network adapters configured on the system. The AdapterCount property can be used to determine the number of adapters that are available.

Mail Message

The Mail Message class can be used to create and process messages in the format defined by the Multipurpose Internet Mail Extensions (MIME) standard. When a message is parsed, it is broken into parts, each consisting of two sections. The first part is called the header section and it describes the format of the data and how it should be represented to the user. The second section is the data itself. A typical mail message without file attachments has one part, with the body of the message being the data. Messages with attachments have multiple parts, each with a header describing the type of data. The class can be then used to extract the data from a multipart message and save it to a file on the local system, delete the part from the message, or add additional parts to the message, such as attaching a file.

The class can also be used to create new multipart messages with alternative content, such a message with both plain text and styled HTML text. Once a message has been created, files can be attached to the message and the application can make any other changes that are needed. The class provides complete access to all headers and content in a multipart message, including the ability to create your own custom headers and make modifications to specific sections.

Initialization

Initialize
Initialize an instance of the class, validating the development license. This method must be called before any properties are changed or any other methods in this class are called by the application.

ComposeMessage
Compose a new message using the specified header field values and content. Using this method, you can create a message with the From, To, Cc and Subject headers already defined, along with any text for the message. You can also optionally provide both plain and styled HTML text versions of the message and the method will automatically create a multipart message.

ClearMessage
Releases the memory allocated for the current message, including any file attachments, and creates a new, empty message.

Uninitialize
Release any resources that have been allocated for the class instance. This method is automatically invoked when the class instance is disposed or goes out of scope.

Message Headers

Each message has one or more headers fields which provide information about the contents of the message. For example, the "From" header field specifies the email address of the person who sent the message. There are a fairly large number of header fields defined by the MIME standard, and applications can also create their own custom headers if they wish. The class gives the application complete access to the header fields in a message. Headers can be examined, modified, created or removed from the message as needed.

GetHeader
This method copies the value of a header field into a string buffer that you provide. To return the value of the common header fields such as "From", "To" and "Subject", you should specify a message part of zero by setting the MessagePart property.

GetFirstHeader
This method returns the value of the first header defined in the current message part, copying it into the string buffer that you provide. This is used in conjunction with the GetNextHeader method to enumerate all of the headers that have been defined.

GetNextHeader

This method returns the value of the next header defined in the current message part. It should be called in a loop until it returns a value of zero (False) which indicates that the last message header has been returned.

SetHeader
Set a message header field to the specified value in the current message part. If the value is an empty string, the message header will be deleted from the message.

DeleteHeader
Delete the specified message header from the current message part.

Message Contents

The content or body of a message contains the text that will be read or processed by the recipient. It may be a simple, plain text message or it may be more complex, such as a combination of plain and styled HTML text or the data for a file attachment. The class provides complete access to the contents of the message, enabling the application to modify, extract, replace or delete specific sections of the message.

Message
This property returns the current message, including the headers and all message parts, as a string. Setting this property will cause the current message to be cleared and replaced by the new value. The string contents must follow the standard specifications for a message. If the property is set to an empty string, the current message is cleared.

Text
This property returns the body of the current message part. Setting this property replaces the entire message body with the new text.

Multipart Messages

Most typical messages contain a single part, which consists of the message headers followed by the contents of the message. However, when files are attached to a message or alternative content types such as HTML are used, a more complex multipart message is required. With a multipart message, the contents of the message are split into logical sections with each section containing a specific part of the message. For example, when a file is attached to a message, one part of the message contains the text to be read by the recipient and another part contains the data for the file.

The first of a multipart message is called part 0, and contains the main header block. This is what defines the headers that you are most familiar with, such as "From", "To" and "Subject". The body of this message part is typically a plain text message that indicates that this is a multipart message. This is done for the benefit of older mail clients that cannot parse MIME messages correctly. Next part, part 1, typically contains the actual body of the message that would be displayed by the mail client. Additional parts may contain file attachments and other information. In the case of a multipart message that contains both plain and styled HTML text versions of a message, part 1 is typically the plain text version of the message while part 2 contains the HTML version. The mail client can then make a decision based on its own configuration as to which version of the message it displays.

Part
This property returns the current message part index. All messages have at least one part, which consists of one or more header fields, followed by the body of the message. The default part, part 0, refers to the main message header and body. If the message contains multiple parts (as with a message that contains one or more attached files), this property can be set to refer to that specific part of the message.

PartCount
This property returns the number of parts in the current message. All messages have at least one part, referenced as part 0. Multipart messages will consist of additional parts which may be accessed by setting the Part property.

CreatePart

Create a new, empty message part. If the message was not originally a multipart message, it will be restructured into one. Otherwise, the new part is simply added to the end of the message. This method will cause the current message part to change to the new part that was just created.

DeletePart
Delete the message part from the message. If the message part is in the middle of the message, it will cause the subsequent parts of the message to be reordered. You should not delete part zero to delete a message; use the DeleteMessage method instead.

Importing and Exporting Messages

The class can be used to import existing messages from a text file and export messages to a text file. Once the message has been parsed, the application can examine or modify specific parts of the message. The following methods are provided to import and export the contents of a message:

ImportMessage
The simplest method of importing a message, this method reads the contents of the specified file and imports it into the current message. Note that the current message contents will be overwritten with the imported message.

ExportMessage
This method exports the current message to a file. When using this method, only certain headers are exported and they may be reordered. To force all headers to be included in the message or to preserve the order of the headers, set the Options property.

File Attachments

In addition to simple text messages, one or more files can be attached to a message. The process of attaching a file involves creating a multipart message, encoding the contents of the file and then including that encoded data in the message. The following methods are provided to manage files attached to the message, as well as attach files to an existing message:

Attachment
A property which returns the name of a file attachment in the current message part. This property serves two purposes, to determine if the current message part contains a file attachment, and if so, what file name should be used when extracting that attachment.

AttachFile
This method attaches the contents of the file to the message. The file will be attached using the specified encoding algorithm and will become the current message part. If the message is not a multipart message, it will be converted to one; if it already is a multipart message, the attachment will be added to the end of the message.

AttachData
This method works in similar fashion to AttachFile, except that instead of the contents of a file, the data in a memory buffer will be attached to the message. If the message is not a multipart message, it will be converted to one; if it already is a multipart message, the attachment will be added to the end of the message.

AttachImage
This method attaches an inline image file to the message. It is similar to the AttachFile method, except that the image is designed to be referenced as an embedded graphic in an HTML message. This method will automatically set the correct header values for an inline image attachment, and enables the developer to specify a content ID which is used in the HTML message.

ExtractFile
Extract the file attachment in the current message part, storing the contents in a file. The attachment will automatically be decoded if necessary. This method also recognizes uuencoded attachments that are embedded directly in the body of the message, rather than using the standard MIME format.

Mail Addresses

The Mail Message class also has methods which are designed to make it easier to work with email addresses. Addresses are typically in the format of "user@domain.com" however additional information can be included with the address, such as the user's name or other comments that aren't part of the address itself. The class can parse these addresses for you, returning them in a format that is suitable for use with other protocols such as the SMTP class.

ParseAddress

Parse an email address that may include an address without a domain name or comments in the address, such as the user's name. For example, the From header field may return an address like "Joe Smith <joe@example.com>"; this method would parse the address and return "joe@example.com", the actual address for the user.

Recipient

It is common for certain headers to contain multiple addresses separated by a comma. These addresses may also include comments such as the user's name. This property array returns a list of valid addresses defined in the current message. For example, the To header field may contain "Tom Jones <tom@example.com>, Jerry Lewis <jerry@example.com>"; this property array would return "tom@example.com" and "jerry@example.com" as the two addresses listed. The total number of addresses that are available is returned by the Recipients property.

Message Storage

The Mail Message class has a collection of methods which makes it simple for an application to store a group of messages together in a single file, search for and retrieve specific message. The collection of messages is referred to as a "message store" and messages may either be stored in a plaintext format or in a compressed binary format.

OpenStore

This method is used to open an existing message store or creates a new storage file. If a storage file has been opened previously, it will be closed and the new storage file will be opened. The storage files may either be plaintext, or stored in a compressed format. It also supports opening storage files in the UNIX mbox format.

StoreSize

This property returns the total number of messages that currently in the message store, including deleted messages. Each message is referred to by an integer which is its index into the storage file.

StoreIndex

This property specifies the current message index into the storage file. Messages are identified by an integer value that starts at one for the first message and increments for each additional message in the storage file. If no message store has been opened, this property will return a value of zero. Changing the value of this property changes the current message index for the message store.

FindMessage

An application can search the message store for messages that match any header value. Searches can be complete or partial, and may be case-sensitive or case-insensitive. For example, this method can be used to enumerate all of the messages in the storage file that were sent by a specific user or match a specific subject.

ReadStore

This method reads a message from the storage file and replaces the current message. If the application modifies the message, it can replace the message in the storage file or discard the changes.

WriteStore

This method writes the current message to the message store. Note that the message store must be opened for write access, and the message will always be appended to the storage file. The StoreIndex property is updated with the index value for the new message.

DeleteMessage
This method flags a message for deletion from the message store. Once a message has been flagged for deletion, it may no longer be accessed by the application. When the storage file is closed, the contents of the deleted message will be removed from the file.

ReplaceMessage
This method replaces an existing message in the storage file, overwriting it with the current message. Unlike many of the other methods which do not permit the application to reference a deleted message, this method can be used to replace a previously deleted message.

CloseStore
The message store must be closed when the application has finished accessing it. This method updates the storage file with any changes, purges all deleted messages and closes the storage file. If the storage file is locked for exclusive access, this method will release that lock, allowing another process to open the file

Network News Transfer Protocol

The Network News Transfer Protocol (NNTP) class enables applications to access a news server, list the available newsgroups, retrieve articles and post new articles. It is common for this class to be used in conjunction with the Mail Message class to construct the articles, since a news article uses the same general format as an email message.

Initialization

Initialize
Initialize an instance of the class, loading the networking library and validating the development license. This method must be called before any properties are changed or any other methods in this class are called by the application.

Connect
Establish a connection to the server. Once the connection has been established, the other methods in the class may be used to access the messages on the server.

Disconnect
Disconnect from the server and release the memory allocated for that client session. After this method is called, the client session is no longer valid.

Reset
Reset the internal state of the component. This can be useful if your application wishes to discard any settings made by a user and return that instance of the class to its default state.

Uninitialize
Unload the networking library and release any resources that have been allocated for the class instance. This method is automatically invoked when the class instance is disposed or goes out of scope.

Newsgroups

News articles are posted in hierarchical groups, similar to how files are stored in folders. Each level in the newsgroup hierarchy is separated by a period, so newsgroup names look like microsoft.public.vc. This is Microsoft's newsgroup for articles about Visual C++ programming. Additional subgroups are used to further narrow the topic; for example, there's the microsoft.public.vc.3rdparty newsgroup for third party tools and components for Visual C++, and the microsoft.public.vc.atl newsgroup which discusses issues related to the Active Template Library. The NNTP class provides the following methods for accessing newsgroups on the server:

GetFirstGroup
This method returns the first available newsgroup on the server. This method is used in conjunction with the GetNextGroup method to enumerate all of the available newsgroups. An overloaded implementation of this method also enables you to specify a date value, where only those newsgroups created on or after that date are returned. This is useful for updating a locally stored list of newsgroups without downloading the complete list of newsgroups each time the client connects to the server.

GetNextGroup
This method returns the next available newsgroup on the server and is used in conjunction with the GetFirstGroup method.

SelectGroup
This method is used to select a newsgroup as the current group. Once selected, the application has access to the articles in that newsgroup.

News Articles

News articles are the messages posted to one or more newsgroups. Articles are referenced by their article number, which is a value assigned by the news server. These articles have a structure that is the same as an email message, with some slightly different headers. Because of this, you can use the Mail Message interface to parse articles that you retrieve, as well as create new articles to post to the server. The following methods are used to access and create news articles:

GetFirstArticle
This method returns information about the first available article in the currently selected newsgroup. This method is used in conjunction with the GetNextArticle method to enumerate all of the available articles in a newsgroup. An overloaded implementation of this method also enables you to specify a starting and ending article number, which enables you to only list those articles within a specific range.

GetNextArticle
This method returns the next available article and is used in conjunction with the GetFirstArticle method.

GetArticle
Retrieve an article from the server, storing the contents in a string buffer or byte array. This can be used to process the contents of an article without the overhead of storing it in a file on the local system.

StoreArticle
Retrieve an article from the server and store it in a file on the local system.

PostArticle
This method posts an article to one or more newsgroups on the server. A newsgroup article is similar to an email message, and the MIME interface may be used to create the article headers and body. One important difference is that the message must contain a header named "Newsgroups" with the value set to the newsgroup or newsgroups that the article should be posted to; multiple newsgroups should be separated by commas. If this header is not defined, the posting will be rejected by the server and the method will return an error. You should also be aware that some servers limit the number of newsgroups that a message can be posted to. When an article is posted to more than one newsgroup at a time, this is called cross-posting. Current convention says that an article should not be cross-posted to more than five newsgroups at a time. Also keep in mind that multiposting (posting the same article to different newsgroups separately) is generally discouraged and should never be done on USENET.

Attaching Files

It is possible to attach files to newsgroup articles; however it should only be done if it is considered appropriate for the group. Many newsgroups have their own acceptable use policies which determine whether or not file attachments, particularly large binary files, are acceptable. If the newsgroup accepts attachments, you can use one of several methods for posting files. It is recommended that you use the File Encoding interface to handle the actual encoding of the data.

Uuencode
A uuencoded file attachment is included directly in the body of the message. Because the MIME interface creates a multipart message even when uuencoding is specified, the File Encoding interface should be used to encode the data and then it should be included in the main body of the message.

Base64
A Base64 file attachment has the same structure as what is used by email messages. This requires that a multipart message be created, with the encoded data attached as a part of the message. You can use the MIME class to create this kind of message. Note that not all third-party newsreaders correctly handle multipart messages.

yEnc
The newest encoding method used on USENET is called yEnc. Similar to uuencoded attachments, the file data is part of the body of the message. The File Encoding class should be used to encode the data and then it should be included in the main body of the message. More information about yEnc encoding can be found at www.yenc.org

Network Time Protocol

The Time protocol control enables an application to retrieve the current time from a server, and optionally synchronize the local system time using that value. The first step that your application must take is to initialize the control. After the control has been initialized, the application can request the current time from a system and update the local system clock if necessary.

Overview

Initialize
Initialize an instance of the class, loading the networking library and validating the development license. This method must be called before any properties are changed or any other methods in this class are called by the application.

GetTime
Return the current time from a server. The time and date retrieved from the server will be returned as a string formatted according to the user's current locale. If the date could not be retrieved, an empty string will be returned.

SetTime
Update the local system time with the value returned by GetTime. This method requires that the current user have the appropriate permissions to modify the system time or the method will fail.

Reset
Reset the internal state of the component. This can be useful if your application wishes to discard any settings made by a user and return that instance of the class to its default state.

Uninitialize
Unload the networking library and release any resources that have been allocated for the class instance. This method is automatically invoked when the class instance is disposed or goes out of scope.

Time Conversion

The class also provides a method which can be used to convert between the local date and time and UTC date and time for the value returned by the server.

ConvertTime
This method enables the application to easily convert between the network time value (which is expressed as a long integer specifying the number of seconds elapsed since midnight, January 1, 1900) and the System.DateTime class.

Post Office Protocol

The Post Office Protocol (POP3) class enables an application to retrieve a user's mail messages and store them on the local system. The control provides support for all of the standard functionality such as listing and downloading messages, as well as extended features such as the ability to retrieve only the headers for a message or just specific header values. The class also has methods for changing the user's password and sending messages if they are supported by the server.

Initialization

Initialize
Initialize an instance of the class, loading the networking library and validating the development license. This method must be called before any properties are changed or any other methods in this class are called by the application.

Connect
Establish a connection to the server. Once the connection has been established, the other methods in the class may be used to access the messages on the server.

Disconnect
Disconnect from the server and release the memory allocated for that client session. After this method is called, the client session is no longer valid.

Reset
Reset the internal state of the component. This can be useful if your application wishes to discard any settings made by a user and return that instance of the class to its default state.

Uninitialize
Unload the networking library and release any resources that have been allocated for the class instance. This method is automatically invoked when the class instance is disposed or goes out of scope.

Managing Messages

There are methods in the POP3 control for managing messages which enables the application to list, delete and retrieve messages stored on the server. Messages are identified by a number, starting with one for the first message in the mailbox. The most typical operation for a POP3 client is to retrieve each message, store it on the local system and then delete the message from the server. Any processing that is done on the message would then be done on the local copy.

Message
This property sets or returns the message number for the currently selected mailbox. Message numbers range from 1 through the number of messages available on the server, as returned by the MessageCount property.

MessageCount, LastMessage
A property which returns the number of messages available for retrieval. There are two values the application should use. One is the number of currently available messages and the other is the last valid message number. As messages are deleted from the server, the total number of available messages will decrease; however, the last available message number will remain constant.

MessageSize
This property returns the size of the message in bytes. One thing to be aware of when using this method is that some servers will only return approximate message sizes. In addition, because of the difference between the end-of-line characters on UNIX and Windows systems, the size reported by the server may not be the actual size of the message when stored on the local system. Therefore, the application should not depend on this value as an absolute. For example, it should not use this value to determine the maximum number of bytes to read from the server; instead, it should read until the server indicates that

the end of the message has been reached.

GetMessage
This method is used to retrieve a message from the server and copy it into a local string or byte array buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled.

StoreMessage
This method downloads a complete message and stores it as a text file on the local system.

DeleteMessage
Mark the message for deletion. When the connection with the server is closed, the message will be removed from the user's inbox. An important difference between the POP3 and IMAP protocols is that when a message is marked as deleted on a POP3 server, that message can no longer be accessed. An attempt to retrieve a message after it has been marked for deletion will result in an error. The only way to undelete a message once it has been deleted is to terminate the connection with the server by calling the Reset method instead of calling the Disconnect method.

Message Headers

The class also includes methods which enable the application to access the headers for a message. This can be useful if the program doesn't want to incur the overhead of downloading the entire message contents.

GetHeader
This method returns the value for a specified header field in the message. Using this method, it is not necessary to download and parse the message header.

GetHeaders
This method retrieves the complete headers for the specified message and stores it in a string or byte array provided by the caller.

MessageUID
This property returns the unique ID (UID) that the server has associated with the message. The UID can be used by an application to track whether or not it has previously viewed the message. Unlike the message number, which can change between client sessions, the message UID is guaranteed to be the same value across sessions until the message is deleted.

Downloading Messages

In some cases, it may be preferable to download a complete message from the server to the local system. This can be easily done with a single method call.

StoreMessage
This method downloads a complete message and stores it as a text file on the local system.

Remote Command Protocol

The Remote Command protocol enables an application to execute commands on a server, with the output of the command returned to the client. The SocketTools control actually implements three related protocols: rexec, rshell and rlogin. The choice of protocols is determined by the port that is selected when a connection is established.

Initialization

Initialize
Initialize an instance of the class, loading the networking library and validating the development license. This method must be called before any properties are changed or any other methods in this class are called by the application.

Reset
Reset the internal state of the component. This can be useful if your application wishes to discard any settings made by a user and return that instance of the class to its default state.

Uninitialize
Unload the networking library and release any resources that have been allocated for the class instance. This method is automatically invoked when the class instance is disposed or goes out of scope.

Executing Commands

When executing commands remotely, there are two similar protocols that can be used. The **rexec** protocol enables a client application to execute a command on a server. Output from the command is returned to the client and the connection is closed when the command terminates. The client connects on port 512 and must provide a user name and password to authenticate the session.

The **rshell** protocol is similar to rexec in that it enables a client to execute a command on a server. Output from the command is returned to the client and the connection is closed when the command terminates. The client connects on port 514 and must provide a user name. The primary difference between the rexec and rshell protocols is that rshell does not require a password. Instead, it uses what is called "host equivalence" to determine if the client is permitted to execute commands as that user. On a UNIX based operating system, host equivalence is controlled by the /etc/hosts.equiv and the .rhosts file in the user's home directory. These files list the host names and user names which are permitted to execute commands using the rshell protocol. Consult your operating system manual pages for more information about how to configure host equivalence.

An important consideration when deciding whether to use rexec or rshell is how the server is configured and the type of command being executed. If there is no entry for the local host in the server's host equivalence tables, then the rexec command should be used instead of rshell.

When using rexec or rshell, it is important to keep in mind that although the command is executed with the privileges of the specified user, that user is not actually logged in. The user's login script is not executed and the program will not inherit the user's normal environment as it would during an interactive session. If you are connecting to a UNIX system, you should not attempt to execute programs which try to put standard input into raw mode; an example of this would be the vi editor. If you are connecting to a Windows system, you should not execute a program which uses a graphical interface. Only programs which read standard input and write to standard output are suitable for use with rexec or rshell.

Execute
Execute the specified command on the server. The rshell or rexec protocol is selected based on the port number that is specified. Output from the command will be returned to the client to be read. When the command terminates, the connection to the server will be closed.

Read

Read the output generated by the command. Your application would typically call this method in a loop until all of the data has been read or an error occurs.

Search
Search for a specific sequence of characters in the output returned by the server. The method returns when the sequence is encountered or when a timeout occurs. The data captured up to the point where the character sequence was matched is returned to the caller for processing.

Remote Login

The **rlogin** protocol is similar to Telnet in that it provides an interactive terminal session. The connection is closed when the user logs out or the shell process on the server is terminated. The client connects on port 513 and must provide a user name and terminal type. If there is an entry in the host equivalence tables for the user and local host, then the client will be automatically logged in and provided with a shell prompt. If there is no host equivalence, the client will be prompted for a password. The terminal emulation control can be used to provide ANSI or DEC VT-220 emulation services if needed.

Login
Establish an interactive login session which is similar to how the Telnet protocol works. If there is no host equivalence with the local host, you will be prompted for a password. Output from the session will be returned to the client, and when the client logs out the connection will be closed.

Read
Reads any output that has been generated by the program executing on the server. Once a connection has been established, users are typically given a command line prompt where they can enter commands to be executed on the server. If the server closes the connection, the Read method will indicate that with an error result and the client can disconnect from the server at that point.

Search
Search for a specific sequence of characters in the output returned by the server. The method returns when the sequence is encountered or when a timeout occurs. The data captured up to the point where the character sequence was matched is returned to the caller for processing.

Write
Send data to the server which will be received as input to the program. In most cases, the server will automatically echo back any characters written as data to be read by the client.

Secure Shell Protocol

The Secure Shell (SSH) protocol enables an application to establish a secure, interactive terminal session with a server, or execute commands remotely on the server, with the output of the command returned to the client. The SocketTools .NET class supports both version 1.0 and 2.0 of the protocol.

Initialization

Initialize

Initialize an instance of the class, loading the networking library and validating the development license. This method must be called before any properties are changed or any other methods in this class are called by the application.

Connect

Establish a connection to the server. Once the connection has been established, the other methods in the class may be used to exchange data with the server.

Disconnect

Disconnect from the server and release the memory allocated for that client session. After this method is called, the client session is no longer valid.

Reset

Reset the internal state of the component. This can be useful if your application wishes to discard any settings made by a user and return that instance of the class to its default state.

Uninitialize

Unload the networking library and release any resources that have been allocated for the class instance. This method is automatically invoked when the class instance is disposed or goes out of scope.

Input and Output

Once connected to the server, any output generated by the command shell or a program executed on the server will be sent as data for the client to read. Any input to the program is sent by the client and received and processed by the server. The following methods are used:

Read

Reads any output that has been generated by the program executing on the server. If the server closes the connection, this method will return zero after all the data has been read. If the method is successful, it will return the actual number of bytes read.

ReadLine

The ReadLine method reads data from the server up to the specified number of bytes or until an end-of-line character sequence is encountered. Unlike the Read method which reads arbitrary bytes of data, this function is specifically designed to return a single line of text data in a string variable.

Peek

The Peek method can be used to examine the data that is available to be read from the internal receive buffer. If there is no data in the receive buffer at that time, a value of zero is returned. The Peek method will never cause the client to block, and so may be safely used with asynchronous connections.

Write

Send data to the server which will be received as input to the program. If the method succeeds, the return value is the number of bytes actually written. This method should always be used when sending binary data to the server.

WriteLine

The WriteLine method sends a line of text to the server and terminates the line with a carriage-return and linefeed control character sequence. Unlike the Write method which writes arbitrary bytes of data to the socket, this method is specifically designed to write a single line of text data from a string.

Command Processing

The SSH protocol can be used to connect to a server, log in and execute one or more commands, process the output from those commands and display it to an end-user using a graphical interface. The user never sees or interacts with the actual terminal session. The class interface provides methods which can simplify this kind of application, reducing the amount of code needed to process the data stream returned by the server.

Execute

This method executes a command on a server and copies the output to a user-specified buffer, with the exit code for the remote program as the method's return value. This is a convenience method that enables you to execute a remote command in a single call, without having to write the code to establish the connection and read the output.

Search

This method is used to search for a specific character or sequence of characters in the data stream returned by the server. The control will accumulate all of the data received up to the point where the character sequence is encountered. This can be used to capture all of the output from a command, or search for specific results returned by the command as it executes on the server.

Simple Mail Transfer Protocol

The Simple Mail Transfer Protocol (SMTP) class enables applications to deliver email messages to one or more recipients. The class provides an interface for addressing and delivering messages, and extended features such as user authentication and delivery status notification. This class is typically used in conjunction with the MailMessage class to create the messages, and the Domain Name Service class to determine what servers are responsible for accepting mail for a specific user.

Mail Exchanges

When a message is delivered to a user, the application must determine what mail server is responsible for accepting messages for that user. This can be accomplished using the Domain Name Services (DNS) protocol, a protocol that is most commonly used to resolve host names such as www.microsoft.com into Internet addresses. This is typically accomplished by sending a request to a nameserver, a computer system that provides domain name services. In addition to resolving host names, nameservers can also provide information about those servers which are responsible for accepting mail for a given domain. There can be multiple servers which process mail for a domain with each server assigned a priority as part of their mail exchange (MX) record. If there is no mail exchange record for a domain, then the domain name itself is used.

To deliver a message directly to the recipient, you must examine the recipient address and request the list of mail exchanges for that user's domain. Using the DNS class, this is done by reading the MailExchange property array. If the recipient address is joe@example.com, you would want to enumerate the mail exchanges for the example.com domain. This will give you the name of the servers that will accept mail for users in that domain. For example, the property may return the host name mail.example.com as the name of the server which will accept mail for users in the example.com domain. Note that it is possible that one or more of the mail exchanges for a domain may not be in the recipient domain itself. In other words, it is possible that smtp.othercorp.net could be returned as a mail exchange for example.com. This is frequently the case when another organization is forwarding mail for that domain.

Therefore, there are four general steps that you must take when delivering mail directly to the recipient:

1. Parse the address of each recipient in the message. If you are using the MailMessage class, the Recipient and Recipients properties can be helpful in extracting all of the recipient addresses. Everything after the atsign (@) in the address is the domain portion of that address.
2. Perform an MX record lookup using the DNS class by setting the HostName property to the domain name and reading the values returned in the MailExchange property array. This property will return the name of the servers responsible for accepting mail for that user. If there are more than one server, they will be returned in order of their relative priority, with the highest priority server having a lower index value. This means that you should attempt to connect to those servers in the order that they are returned by the property, starting with an index value of zero.
3. Attempt to connect to the first server returned by the MailExchange property array. The connection should be on the default port, and you should not attempt to use any authentication. If the server accepts the connection, then use the SendMessage method to deliver the message. If the connection is rejected or the message is not accepted, attempt to connect to the next mail exchange server until all servers have been tried.
4. If no mail exchange servers were returned by the MailMessage class MailExchange property, or you could not connect to any of them, attempt to connect to the domain specified in the address using the default port. If the connection succeeds, then deliver the message. If you cannot connect or the message is not accepted, then report to the user

that the message could not be delivered.

One last important consideration is that many Internet Service Providers now block outbound connections on port 25 to any mail servers other than their own. If you are unable to establish any connections, either with the error that the connection was refused or it consistently times out, contact your ISP to determine if port 25 is being blocked as an anti-spam measure. If this is the case, it will be required that you relay all messages through their mail servers.

Relay Servers

In some situations it may not be possible to send mail directly to the server that accepts mail for a given domain. The two most common situations are corporate networks which have centralized servers that are responsible for delivering and forwarding messages, or an Internet Service Provider (ISP) which specifically blocks access to all mail servers other than their own. This is usually done as either a security measure or as a means to inhibit users from sending unsolicited commercial email messages. If the standard SMTP port is being blocked, then any connection attempts will either fail immediately with an error that the server is unreachable, or the connections will simply time-out. In either case, a relay server must be specified in order to send email messages.

A relay server is a system which will accept messages addressed to users who may be in a different domain, and will relay those messages to the appropriate server that does accept mail for the domain. Using a relay server is generally easier than sending messages directly to the recipient. In order to send a message through a relay, you need to perform the following steps:

1. Connect to the relay server as you would normally.
2. Authenticate the client to the server. This may or may not be required, depending on how the server is configured. Some servers may be configured to only require authentication if you are connecting from an IP address that is not recognized as part of that system's network, for example, if you are connecting using a different Internet Service Provider. Others may always require authentication. Check with the server administrator if necessary to determine if and when authentication is required.
3. Use the SendMessage method to deliver the message to the recipients through the relay server. If there are multiple recipients, you can use the MailMessage class to enumerate the recipient addresses and then pass them to the SendMessage method.

It is important to note that using a mail server as a relay without the permission of the organization or individual who owns that server may violate Acceptable Use Policies and/or Terms of Service agreements with your service provider. Systems which relay messages from anyone, regardless of whether the message is coming from a recognized domain, are called open relays. Because open relays are often used to send unsolicited email, many administrators block mail that comes from one. It is recommended that users check with their network administrators or Internet service providers to determine if access to external mail servers is restricted and what is the acceptable use policy for relaying messages through their mail servers.

Initialization

Initialize

Initialize an instance of the class, loading the networking library and validating the development license. This method must be called before any properties are changed or any other methods in this class are called by the application.

Connect

Connect to the server, using either a host name or IP address. This method creates the client session and must be called before your application attempts to request a resource from the server.

Authenticate

Authenticate yourself to the server using a username and password. This method should be called immediately after the connection has been established to the server. This is typically required if you are attempting to use the mail server as a relay, asking it to forward the message on to the server that actually accepts email for the recipient. Many Internet Service Providers (ISPs) require that users authenticate prior to sending mail through their servers. You may need to contact the server administrator to determine if authentication is required.

Disconnect
Disconnect from the server and release the memory allocated for that client session. After this method is called, the client session is no longer valid.

Reset
Reset the internal state of the component. This can be useful if your application wishes to discard any settings made by a user and return that instance of the class to its default state.

Uninitialize
Unload the networking library and release any resources that have been allocated for the class instance. This method is automatically invoked when the class instance is disposed or goes out of scope.

Message Delivery

There are two general methods that can be used to deliver messages through the mail server. In most cases, it can be done with a single method call. However, there are some circumstances where it would be more appropriate to perform the transaction in stages. The SMTP class supports both methods.

SendMessage
This is the simplest method for sending an email message through the server. You provide the sender and recipient addresses, along with the message contents and the method will submit the message to the server for delivery.

CreateMessage
This method begins a transaction in which a message is dynamically composed, addressed and delivered in stages. You provide the sender address and message size to this method, and after it returns you begin the next stage, which is addressing the message.

AddRecipient
This method adds a recipient address to the recipient list for the message. This should be called once for each recipient, as well as for any recipients who are to receive "blind copies" of the message. A blind copy is when the message is sent to a recipient, but that recipient's address is not listed in any of the headers of the message; the other recipients will be unaware that the message was delivered to him. Most servers have a limit of approximately 100 recipients per message. It is possible that this method will return an error for a specific recipient address; the address may be malformed or it may not be acceptable for some other reason. This does not mean that the message will be rejected in its entirety, only that the specified recipient is not acceptable.

AppendMessage
This method should be called after all of the recipients have been added. It is used to send the contents of the message to the server. It is also possible to use the lower level Write method to send data directly to the server, however AppendMessage is generally easier to use and can write data from memory, the system clipboard or from a file on disk.

CloseMessage
This method is called after the entire message has been sent to the server. This terminates the transaction and the message is submitted for delivery. Note that it is possible for the server to accept the message up to this point and then reject it at this final step due to some restriction, such as the message being too large.

Windows Sockets (SocketWrench)

The SocketWrench class provides a simplified interface to the Windows Sockets API. It was designed to be easier to use, and to provide properties and methods which eliminate much of the redundant coding common to network programming. SocketWrench also supports creating client and server applications which use the SSL and TLS security protocols without any dependencies on third-party security libraries.

Initialization

Initialize

Initialize an instance of the class, loading the networking library and validating the development license. This method must be called before any properties are changed or any other methods in this class are called by the application.

Connect

Connect to the server, using either a host name or IP address. When an application calls this method, it will be acting as a client. This method creates the socket and must be called before your application attempts to exchange data with a server. For an asynchronous session, set the Blocking property to False.

Listen

Begin listening for incoming client connections. When an application calls this method, it will be acting as a server. Once the Listen method returns, the socket is created and that socket handle is used by the Accept method accept an incoming client connection. For an asynchronous session, set the Blocking property to False.

Accept

Accept a connection from a client. This method should only be called if the application has previously called the Listen method. If there is no client waiting to connect at the time this method is called, it will block until a client connects or the timeout period is reached.

Uninitialize

Unload the networking library and release any resources that have been allocated for the class instance. This method is automatically invoked when the class instance is disposed or goes out of scope.

Input and Output

When a TCP connection is established, data is sent and received as a stream of bytes. The following methods can be used to send and receive data over the socket:

Read

A low-level method used to read data from the socket and copy it to the string buffer or byte array provided by the caller. If the server closes the connection, this method will return zero after all the data has been read. If the method is successful, it will return the actual number of bytes read. This method should always be used when reading binary data from the server into a byte array.

ReadLine

Read a line of text from the socket, up to an end-of-line character sequence or when the server closes the connection. This method is useful when the client and server are exchanging textual data, as is common with most command/response application protocols.

ReadStream

A high-level method used to read a stream of bytes and copy it to a string buffer or byte array provided by the caller. This method can be used to read an arbitrarily large amount of data in a single call.

Write

A low-level method used to write data to the socket. If the method succeeds, the return value is the number of bytes actually written. This method should always be used when sending binary data to the remote host.

WriteLine
Write a line of text to the socket, terminating it with an end-of-line character sequence. This method is useful when the client and server are exchanging textual data, as is common with most command/response application protocols.

WriteStream
A high-level method used to write a stream of bytes to the socket. This method can be used to write an arbitrarily large amount of data to the socket in a single call.

IsReadable
This property is used to determine if there is data available to be read from the socket. If the property returns a value of True, the Read method will return without causing the application to block. If the property returns False, there is no data available to read from the socket.

IsWritable
This property is used to determine if data can be written to the socket. In most cases this will return True, unless the internal socket buffers are full.

Host Name Resolution

The class can be used to resolve host names into IP addresses, as well as perform reverse DNS lookups converting IP addresses into the host names that are assigned to them. The class will search the local system's host table first, and then perform a nameserver query if required.

HostAddress
This property can be used to set the IP address for a server that you wish to communicate with. If the address is valid and matches an entry in the host table, the HostName property will be changed to match the address.

HostName
This property should be set to the name of the server that you wish to communicate with. If the name is found in the host table, the HostAddress property is updated to reflect the IP address of the host. Note that it is legal to assign an IP address to this property, but it is not legal to assign a host name to the HostAddress property.

Local Host Information

Several methods are provided to return information about the local host, including its fully qualified domain name, local IP address and the physical MAC address of the primary network adapter.

LocalName
Return the fully qualified domain name of the local host, if it has been configured. If the system has not been configured with a domain name, then the machine name is returned instead.

LocalAddress
Return the IP address of the local host. If a connection has been established, then the IP address of the network adapter that was used to establish the connection will be returned. This can be particularly useful for multihomed systems that have more than one adapter and the application needs to know which adapter is being used for the connection.

ExternalAddress
Return the IP address assigned to the router that connects the local host to the Internet. This is typically used by an application executing on a system in a local network that uses a router which performs Network Address Translation (NAT).

PhysicalAddress
Return the physical MAC address for the primary network adapter on the local system.

AdapterAddress
This property array returns the IP addresses that are associated with the local network or remote dial-up

network adapters configured on the system. The AdapterCount property can be used to determine the number of adapters that are available.

Telnet Protocol

The Telnet Protocol control enables an application to connect to a Telnet server, which provides an interactive terminal session similar to how character based consoles and terminals work. The user can login, enter commands and interact with applications programmatically or in conjunction with the terminal emulation control.

Initialization

Initialize
Initialize an instance of the class, loading the networking library and validating the development license. This method must be called before any properties are changed or any other methods in this class are called by the application.

Connect
Establish a connection to the server. Once the connection has been established, the other methods in the class may be used to exchange data with the server.

Disconnect
Disconnect from the server and release the memory allocated for that client session. After this method is called, the client session is no longer valid.

Reset
Reset the internal state of the component. This can be useful if your application wishes to discard any settings made by a user and return that instance of the class to its default state.

Uninitialize
Unload the networking library and release any resources that have been allocated for the class instance. This method is automatically invoked when the class instance is disposed or goes out of scope.

Input and Output

Once connected to the Telnet server, any output generated by a program on the server will be sent as data for the client to read. Any input to the program is sent by the client and received and processed by the server. The following methods are used:

Read
Reads any output that has been generated by the program executing on the server. When the client first connects, the server typically executes a login program that requests the users authenticate themselves by entering a user name and password. Once the user has logged in, they are usually given a command line prompt where they can enter commands to be executed on the server. If the server closes the connection, the Read method will indicate that with an error result and the client can disconnect from the server at that point.

Write
Send data to the Telnet server which will be received as input to the program. If the local echo option is enabled, then the client is also responsible for writing the input data to the display device, if there is one. If local echo is not enabled, the server will automatically echo back any characters written as data to be read by the client.

Local Echo

Telnet supports a mode of operation where the it is the responsibility of the client to echo any data sent to the server. This is controlled by the LocalEcho property.

LocalEcho
If this property is set to True, it is the responsibility of the client to echo any data that it is sending to the server. For example, if the character "A" is sent to the server, the application must also send the character

"A" to whatever interface the user is interacting with, such as a terminal emulation window. The default mode is for this option to be disabled, which means that the server will echo back any data that is sent to it.

Command Processing

The Telnet protocol can be used to connect to a server, log in and execute one or more commands, process the output from those commands and display it to an end-user using a graphical interface. The user never sees or interacts with the actual terminal session. The Telnet interface provides methods which can simplify this kind of application, reducing the amount of code needed to process the data stream returned by the server.

Login
This method is used to automatically log a user in, using the specific user name and password. This method is specifically designed for UNIX based servers or Windows servers which emulate the same basic login sequence.

Search
This method is used to search for a specific character or sequence of characters in the data stream returned by the server. The control will accumulate all of the data received up to the point where the character sequence is encountered. This can be used to capture all of the output from a command, or search for specific results returned by the command as it executes on the server.

Terminal Emulation

The Terminal Emulation control provides a virtual terminal interface for emulating an ANSI or DEC VT-220 compatible character-based terminal. It can be used in conjunction with the Telnet interface or the Remote Command interface to display the output of commands executed on a server. It can also be used independently of any other networking control, such as providing emulation services for a serial connection.

Display Management

The control provides a number of properties and methods to manage and update the virtual display. The most commonly used methods are:

BackColor
This property can be used to change the background color displayed by the virtual terminal.

ColorMap
This property array can be used to change the default colors which are used when escape sequences are used to change the foreground or background color of a character cell. In most cases the default color map will be appropriate, but applications can change the RGB values associated with an entry in the color map if needed. For example, the default value for the color gray is at position 8 in the color map index with an RGB value of 192,192,192. If you wanted to use a darker color, you could change the RGB value to 128,128,128

Emulation
This property specifies the type of emulation that will be performed by the control. The control is capable of emulating an ANSI console, a DEC VT-100 and DEC VT-220/320 terminal.

Font
This property sets the font which is used by the control to draw text on the display window. It is recommended that you only used fixed-width fonts such as Terminal or Courier New.

ForeColor
This property can be used to change the foreground color displayed by the virtual terminal.

Write
This is the most commonly used method of writing to the display. This method will automatically parse the data being written for escape sequences and update the display appropriately.

Refresh
Refresh the virtual display, updating the current cursor position and caret. The control will periodically refresh the display automatically based on its own internal state, but the application can call this if it wishes to force the display to refresh at that time.

Reset
This method can be used to reset the display window, the font being used and the size of the display. Note that resetting the display causes the contents of the display to be cleared.

Cursor Control

There are a number of properties and methods which enable an application to have direct control over cursor positioning, clearing the display and so on. In most cases these methods are called automatically by the control as the result of processing the escape sequences found in the data being written to the display. However, an application can choose to manage the display itself. One important thing to keep in mind is that the X,Y positions used by these properties and methods refer to the cursor position in the virtual display and correspond to columns and rows, not pixels.

There is also a slight difference in terminology that you should be aware of when reading the technical reference documentation. In Windows, the term "cursor" is typically used to refer to the mouse pointer, while "caret" is used to refer to the blinking marker that is displayed at the current position in the display. In the documentation for the emulator, the term "cursor" is used in the same way that it is used for character based terminals, as the marker for the current position in the display. Therefore, in terms of the control, you can think of the cursor and the caret as being synonymous.

CursorX
This property returns the current position of the cursor in the display, or can be used to change the current position. The current position is given in columns and indicates where the next text character will be displayed.

CursorY
This property returns the current position of the cursor in the display, or can be used to change the current position. The current position is given in rows and indicates where the next text character will be displayed.

Clear
This method clears the contents of the display. You can clear from the start of the display to the current cursor position, from the current position to the end of the display or the entire display.

DelLine
This method deletes the line at the current cursor position, shifting the remaining lines in the display up.

InsLine
This method inserts a blank line at the current cursor position, shifting the following lines down.

ScrollDown
This method scrolls the display down by one line.

ScrollUp
This method scrolls the display up by one line.

Function Key Mapping

Another aspect of terminal emulation is how function keys and other special keys are handled by the application. The emulation control can be used to convert Windows virtual key codes into the escape sequences that are generated by character based terminals.

KeyMap
This property array allows the application to define character sequences that should be mapped to special keys. When a special key is pressed in the emulation window and there is an entry for it in the key map, the KeyMapped event is fired. For example, if the user presses the F1 key on the keyboard, the control will translate that key code into the three characters escape sequence ESC O P (the ASCII codes 27, 79, 80). That sequence of characters should be sent to the server, which will recognize it as the F1 function key being pressed. It is important to note that the different emulation types have different key mappings. Therefore, the server must be set to recognize the same type of terminal that you are emulating. If you have the emulation set as VT-220 but the server thinks that you are emulating a VT-100, it will not recognize some of the escape sequences correctly.

KeyMapped
This event is generated when the user presses a special key while the emulation window has focus, and that key is mapped to a string using the KeyMap property array. Typically an application will use this event to send the mapped key escape sequence to a server, such as a Telnet server.

Web Location Class

The SocketTools.WebLocation class enables an application to obtain geographical information about the physical location of the computer system based on its external IP address. This can allow developers to know where their application is being used, and provide convenience functionality such as automatically completing a form based on the location of the user.

The connection to the location service is always secure and does not require you subscribe to any third-party services. The accuracy of this information can vary depending on the location, with the most detailed information being available for North America. The country and time zone information for all locations is generally accurate. However, as the location information becomes more precise, details such as city names, postal codes and specific geographic locations (e.g.: longitude and latitude) may have reduced accuracy.

Software which is designed to protect the privacy of users, such as those which route all Internet traffic through proxy servers or VPNs, can significantly impact the accuracy of this information. In this case, the data returned in this structure may reflect the location of the network or proxy server, and not the location of the person using your application. It is recommended you always request permission from the user before acquiring their location, have them confirm the location is correct and provide a mechanism for them to update the information.

Methods

To obtain the location of the local computer system, use the following methods:

Initialize
Initialize an instance of the class, loading the networking library and validating the development license. This method must be called before any properties are changed or any other methods in this class are called by the application.

Update
This method causes the class to update its various properties with information about the current location. The location service is queried to obtain current information about the physical location of the computer system based on its external IP address. The location data is cached and additional queries are only performed if it detects the external IP address for the local system has changed.

Uninitialize
Unload the networking library and release any resources that have been allocated for the class instance. This method is automatically invoked when the class instance is disposed or goes out of scope.

Properties

The following properties provide information about the current location of the local computer:

Property	Description
ASNumber	An integer which is used to uniquely identify a global network (autonomous system) which is connected to the Internet. This value can be used to determine the ownership of a particular network.
CityName	A string which identifies the city at this location. These names will always be in English, regardless of the current system locale. If the city name cannot be determined, this member may contain an empty string.
Coordinates	A string which specifies the location expressed using the Universal Transverse Mercator (UTM) coordinate system with the WGS-84 ellipsoid. These coordinates are commonly used with the Global Positioning System (GPS).
CountryAlpha	A string which contains the ISO 3166-1 alpha-2 code assigned to the country. For example, the alpha-2 code for the United States is "US".

CountryCode	An integer value which identifies the country using the standard UN country code. For example, the numeric country code for the United States is 840.
CountryName	A string which contains the full name of the country in which the external IP address is located, such as "United States". These names will always be in English, regardless of the current system locale.
IPAddress	A string which contains the external IP address for the local system. If the system has been assigned multiple IP addresses, it reflects the address of the interface used to establish the connection with the location server.
Latitude	A real number which specifies the latitude of the location in decimal format. A positive value indicates a location which is north of the equator, while a negative value is a location which is south of the equator.
LocalTime	The current date and time at the location, adjusted for its time zone and whether or not it's in daylight savings time.
LocationId	A string which contains contains a string of hexadecimal characters which uniquely identifies the location for this computer system. This value is used internally by the location service, and may also be used by the application for its own purposes.
Longitude	A real number which specifies the longitude of the location in decimal format. A positive value indicates a location which is east of the prime meridian, while a negative value is a location which is west of the prime meridian.
Organization	A string which identifies the organization associated with the local system's external IP address. For residential end-users this is typically the name of their Internet Service provider, however it may also identify a private company.
PostalCode	A string which contains the postal code associated with the location. In the United States, this is a 5-digit numeric code. Local delivery portions of a postal code (such as the ZIP+4 code in the United States) are not included.
RegionCode	An integer which identifies the geographical region. This value corresponds to standard UN M49 region codes.
RegionName	A string which identifies a broad geographical area, such as "North America" or "Southeast Asia".
Subdivision	A string which identifies a geopolitical subdivision within a country. In the United States, this will contain the full name of the state or commonwealth. In Canada, this will contain the name of the province or territory.
SubdivisionCode	A string which is either a two- or three-letter code which identifies a geopolitical subdivision within the country. These codes are defined by the ISO 3166-2 standard. For example, the code for the state of California in the United States is "CA".
Timezone	A string which specifies the full time zone name. These names are defined by the Internet Assigned Numbers Authority (IANA) and have values like "America/Los_Angeles" and "Europe/London".
TzOffset	A integer which specifies the number of seconds east or west of the prime meridian (UTC). A positive value indicates a time zone which is east of the prime meridian and a negative value indicates a time zone which is west of the prime meridian.
TzShortName	A string which specifies the abbreviated time zone code. If daylight savings time is used within the time zone, then this value can change based on whether or not daylight savings is in effect. If a short time zone code cannot be determined, a value such as "UTC+9" may be returned, indicating the number of hours ahead or behind UTC.

Web Storage Class

The SocketTools.WebStorage class provides private cloud storage for uploading and downloading shared data files which are available to your application. This is primarily intended for use by developers to store configuration information and other data generated by the application. For example, you may want to store certain application settings, and the next time a user or organization installs your software, those settings can be downloaded and restored.

The connection to the storage service is always secure, using TLS 1.2 and AES-256 bit encryption. There are no third-party services you need to subscribe to, and there are no additional usernames or passwords for you to manage. Access to the service is associated with an account which is created when you purchase a development license, and the security tokens are bound to the runtime license key used when initializing the API. You also have the option to compress and encrypt your data you store using the FileEncoder class.

Terminology

When you get started with the WebStorage class, you'll notice there is some different terminology which is used. This will provide an overview of that terminology, and compare it to common terms used with traditional protocols like FTP. When accessing an FTP server, you generally deal with ***directories***, ***files***, ***names*** and ***types*** (generally whether the file is binary or text). The storage control has similar concepts, but uses somewhat different terminology.

Application Identifiers

An application identifier (AppId) is a null terminated string which uniquely identifies your application. This string, used in conjunction with your runtime license key, is used to generate an access token. This token is used to access the storage container which contains the data which you've stored.

It is recommended you use a standard format for the AppId which consists of your company name, application name and optionally a version number. Some examples of an AppId string would be:

- MyCompany.MyApplication
- MyCompany.MyApplication.1

It is important to note with these two example IDs, although they are similar, they reference two different applications. Objects stored using the first ID will not be accessible using the second ID. If you want to store objects which should be shared between all versions of the application, it is recommended you use the first form, without the version number. If you want to store objects which should only be accessible to a specific version of your application, then it is recommended you use the second form which includes the version number.

The AppId must only consist of ASCII letters, numbers, the period and underscore character. Whitespace characters and non-ASCII Unicode characters are not permitted. The maximum length of the string is 63 characters. It is not required for your application to create a unique AppId. Each storage account has a default internal AppId named `SocketTools.Storage.Default`. This AppId is used if a NULL pointer or an empty string is specified.

Containers

Storage containers are somewhat analogous to directories or folders in a file system, however they are general purpose and designed to allow you to control how your application accesses the data that's been stored. There are four container types which are defined by the control, and you can think of them as types of boxes or file cabinets which you store your data in.

It is important to keep in mind these containers are available to all users of your application, your program controls who has access to any particular data file. Your users will not be able to "browse" any of the containers unless you specifically provide that capability by implementing it in your own code. There is no public access to any of the data which you upload, and our service does not use an open API accessible by third parties.

The storage containers are identified using the WebStorage.Container enumeration:

`Container.storageGlobal`
The global storage container which is available to all users of your application. Any data stored in this container is available to everyone who uses your software. Unless you have a specific need to limit access to the data to a specific user or group of users, this is the recommended container you use to store data.

`Container.storageDomain`
The domain storage container is limited to users in the same local domain, defined either by the name of the domain or workgroup assigned to the computer system. This can provide a kind of organization wide storage, but it does depend on the domain being unique. For example, if you are using domain storage for your application, and you have multiple customers who have systems part of the default "Workgroup" domain, they would all share the same container. If the domain or workgroup name changes, then data stored in the container would no longer be available.

`Container.storageMachine`
The local machine storage container is associated with the physical computer system your application is running on. The machine is identified by unique characteristics of the system, including the boot volume GUID. Data stored in this container can only be accessed from the application running on that particular system. If the operating system is reinstalled, the machine ID will change and data stored in this container would no longer be available.

`Container.storageUser`
The current user storage container is associated with the current user who is using your application. The user identifier is based on the Windows Security Identifier (SID) assigned to the account when it's created. If the user account is deleted, the data stored in this container will no longer be available to the application. Another user on the same computer system would not be able to access the data in this container.

If you decide to use anything other than global storage, the data your application stores can be orphaned if the system configuration or user account changes. It's recommended you store critical application data and general configuration information using `Container.storageGlobal` and use other non-global storage containers for configuration information which is unique to that system and/or user which is not critical and can be easily recreated. If you're concerned about protecting the data you upload to global storage, you can encrypt it prior to storing it.

Objects

Storage objects are similar to files in a file system. They are discrete blocks of data, associated with a label (name), have attributes and are associated with a particular content type. However, an object does not need to be an actual file on the local system. For example, you could store an object which is a string, a pointer to a structure, or any block of memory. You could also just store a complete file as an object. Unlike files, you cannot perform partial reads of an object or "seek" into certain parts of a stored object. Of course, you can download an object, either in memory or to a local file, and perform whatever operations you require on the data.

Labels

Object labels are similar to file names, and are a way to identify a stored object instead of using its internal object ID. However, there are some important differences. The most significant difference being labels are case-sensitive, unlike Windows file names. An object with the label "AppConfig" is considered to be different than one with the label "appconfig". Labels can contain Unicode characters, but they cannot contain control characters.

You can also use forward slashes or backslash characters in the label, but it's important to note objects are not stored in a hierarchical structure. Your application can store objects using a folder-like structure, but it's not something which is enforced by the API.

Media Type

Each object your application stores is associated with a media type (also called a content type) which identifies the object's data. This uses the standard MIME media type designations, such as "text/plain" or "application/octet-stream". Your application can explicitly specify the media type you want to associate with the object, or you can have the API choose for you, based on the contents of the object and using the label as a hint for what it may contain. For example, if you create an object with the label "AppConfig.xml" and it contains text, then the API will select "text/xml" as the default media type.

Initialization

The first step your application must take is to initialize the control and then open a storage container. The following methods are available for use by your application:

Initialize
nitialize an instance of the class, loading the networking library and validating the development license. This method must be called before any properties are changed or any other methods in this class are called by the application.

Open
Opens a storage container for your application. Subsequent operations, such as storing, retrieving and copying objects will be performed within this container.

Close
Close the storage container and release the resources allocated for the session.

Uninitialize
Unload the networking library and release any resources that have been allocated for the class instance. This method is automatically invoked when the class instance is disposed or goes out of scope.

Data Storage

The control provides methods to upload and download to the storage container. You can store the contents of local files, or you can create objects from memory using strings or byte arrays.

GetData
Download object data and store it in a string or byte array provided by the caller.

GetFile
Download object data and store it in a file on the local system.

PutData
Upload object data in a string or byte array and store it as an object in the current container. This function would typically be used to store binary data, including compressed or encrypted text.

PutFile
Upload the the contents of a local file and store it as an object in the current container.

Data Management

The data management methods allow you to obtain information about stored objects and perform typical operations such as copying, renaming and deleting objects from the container.

FindFirst
Enables your application to search for and enumerate objects in a container based on their label and/or

their media type. This method is used in conjunction with the FindNext method to list all matching objects in a container.

CompareData
Compares the contents of a string or byte array with the data stored in an object. This method can be used to determine if the contents of the buffer have changed since the data was previously stored using the PutData method.

CompareFile
Compares the contents of a local file with the data in a stored object. This method can be used to determine if the contents of a file have changed since it was previously stored using the PutFile method.

Copy
Copies the contents of a stored object to a new container, or duplicating the object within the same container using a different label.

Move
Moves the contents of a stored object to a new container.

Rename
Changes the label associated with a stored object. The new label for the object cannot already exist in the same container. If you want to change the label to one already assigned to an existing object, the object must first be deleted.

Delete
Removes the stored object from the container. This operation is immediate and permanent. Deleted objects cannot be recovered by the application at a later time.

DeleteAll
Deletes all objects which are stored in the current open container. This method resets the container back to its initial state, deleting all object metadata from the database and removing all stored data. This operation is immediate and the objects stored in the container are permanently deleted. They cannot be recovered by your application.

Other Methods

Several additional methods are available, allowing your application register and de-register custom application identifiers and validate object labels.

RegisterId
Register a new application identifier (AppId) to be used to access a storage container. It is not required you create a unique application ID, but it can be helpful to distinguish stored content between different versions of your applications.

UnregisterId
Unregister an application identifier which was previously registered by your application. You should be extremely careful when using this function because it permanently delete all stored objects created using the AppId value. Internally it revokes the access token granted to your application and causes the server to expunge all objects in the container associated with the token.

ValidateId
A method which can be used to validate an application identifier, ensuring it is valid and has been registered.

ValidateLabel
A method which can be used to validate an object label to ensure it does not contain any invalid characters. This would be primarily used by applications which allow a user to specify the label names for the objects being stored.

Whois Protocol

The Whois protocol class provides an interface for requesting information about an Internet domain name. When a domain name is registered, the organization that registers the domain must provide certain contact information along with technical information such as the primary name servers for that domain. The Whois protocol enables an application to query a server that provides that registration information. The Whois class provides an interface for requesting that information and returning it to the program so it can be displayed or processed.

Overview

Initialize

Initialize an instance of the class, loading the networking library and validating the development license. This method must be called before any properties are changed or any other methods in this class are called by the application.

Connect

Connect to the server, using either a host name or IP address. Once the connection has been established, the other methods in the class may be used to retrieve information from the server.

Search

This method submits a search keyword to the server. The keyword may specify a domain name, a user handle or a user mailbox, depending on the search type. Note that not all servers support all search types. For example, many servers no longer support searching for user information based on email addresses.

Read

Read the data returned by the server, storing it in a string variable or byte array that is specified by the caller. This will contain the information about the domain specified when the Search method was called. Note that the data returned will typically be text, however it may not follow the same end-of-line conventions as Windows. For example, if the server is a UNIX or Linux system, the end-of-line may be indicated by a single linefeed, rather than a carriage-return/linefeed pair. Your application will have to account for this if the data is being displayed as-is to a user.

Disconnect

Disconnect from the server and release the memory allocated for that client session. After this method is called, the client session is no longer valid.

Reset

Reset the internal state of the component. This can be useful if your application wishes to discard any settings made by a user and return that instance of the class to its default state.

Uninitialize

Unload the networking library and release any resources that have been allocated for the class instance. This method is automatically invoked when the class instance is disposed or goes out of scope.

SocketTools Namespace

Classes

Class	Description
DnsClient	The **DnsClient** class resolves domain names into Internet addresses and return information about a remote host, such as the servers that are responsible for accepting mail for the domain.
FileEncoder	The **FileEncoder** class encodes and decodes files using standard algorithms such as base64, uuencode and quoted-printable. The class can also be used to compress and expand files, as well as encrypt or decrypt file data using AES encryption.
FtpClient	The **FtpClient** class implements the File Transfer Protocol (FTP) o transfer files between the local system and a remote server. This class supports both high level operations, such as uploading or downloading files, as well as a collection of lower-level file I/O functions. In addition to file transfers, an application can create, rename and delete files and directories, search for files using wildcards and perform other common file management functions. Secure file transfers are supported using TLS 1.2 and SSH 2.0.
FtpServer	The **FtpServer** class implements an embedded, lightweight server that can be used to exchange files with a client using the standard File Transfer Protocol. The server can accept connections from any third-party application or a program developed using the SocketTools.FtpClient class. Secure file transfers are supported using TLS 1.2.
HttpClient	The **HttpClient** class implements the Hypertext Transfer Protocol (HTTP), a lightweight, stateless application protocol used to access resources on web servers, as well as send data to those servers for processing. The class provides direct, low-level access to the server and the commands that are used to retrieve resources (i.e.: documents, images, etc.). The class also provides a simple interface for downloading resources to the local host, similar to how the SocketTools.FtpClient class can be used to download files. This class supports secure connections using TLS 1.2 and authentication using OAuth 2.0 bearer tokens.
HttpServer	The **HttpServer** class implements an embedded, lightweight server that can be used to provide access to documents and other resources using the Hypertext Transfer Protocol (HTTP). The server can accept connections from any standard web browser, third-party applications or programs developed using the SocketTools.HttpClient class. Secure connections are supported using TLS 1.2.
IcmpClient	The **IcmpClient** class implements the Internet Control Message Protocol (ICMP) which can be used to determine if a remote host is reachable and how packets of data are routed to that system. This class can be used to check if a system is reachable and the amount of time that it takes for a packet of data to make a round trip from the local system, to the remote host and then back again. It can also trace the route that a packet of data takes from the local system to the remote host, and can be used to identify potential problems with overall throughput and latency.

ImapClient	The **ImapClient** class implements the Internet Message Access Protocol (IMAP) is an application protocol which is used to access a user's email messages which are stored on a mail server. This class implements the current standard for this protocol, and provides functions to retrieve messages, or just certain parts of a message, create and manage mailboxes, search for specific messages based on certain criteria and so on. This class is typically used in conjunction with the SocketTools.MailMessage class which is used to process the messages that are retrieved from the server. This class supports secure connections using TLS 1.2 and authentication using OAuth 2.0 bearer tokens.
InternetDialer	The **InternetDialer** class provides a way for client applications to connect to the Internet using Microsoft Windows Remote Access Services (RAS). To use this class, the dial-up networking software must be installed on the local system. For access to the Internet, the TCP/IP protocol must be installed and configured. The class may configured to use either the SLIP or PPP protocols, depending on the requirements of the service provider. Refer to your system documentation for information about installing and configuring dial-up networking on your system.
InternetServer	The **InternetServer** class provides a simplified interface for creating event-driven, multithreaded server applications using the TCP/IP protocol. The interface is similar to the SocketTools.SocketWrench class, however it is designed specifically to make it easier to implement a server application without requiring the need to manage multiple socket classes. In addition, the class supports secure connections using TLS 1.2.
MailMessage	The **MailMessage** class provides an interface for composing and processing email messages and newsgroup articles which are structured according to the Multipurpose Internet Mail Extensions (MIME) standard. Using this class, an application can easily create complex messages which include multiple alternative content types, such as plain text and styled HTML text, file attachments and customized headers. This class is typically used in conjunction with the SocketTools.ImapClient and SocketTools.PopClient classes.
NetworkTime	The **NetworkTime** class provides an interface for synchronizing the local system's time and date with that of a server. The time values returned are in in Coordinated Universal Time and be adjusted for the local host's time zone. The class enables developers to query a server for the current time and then update the system clock if desired.
NewsFeed	The **NewsFeed** class provides an interface parsing Really Simple Syndication (RSS) feeds. A news feed is published in XML format, which contains one or more items that includes summary text, hyperlinks to source content and additional metadata that is used to describe the item. News feeds can be used for a variety of purposes, including providing updates for weblogs, news headlines, video and audio content. News feeds can be accessed remotely from a web server, or locally as an XML formatted text file.
NntpClient	The **NntpClient** class implements the Network News Transfer Protocol (NNTP) which is used to download and post news articles. This is similar in functionality to bulletin boards or message boards, where topics are organized hierarchically into groups, called newsgroups. The largest

	collection of public newsgroups available is called USENET, a world-wide distributed discussion system. Secure connections are supported using TLS 1.2.
PopClient	The **PopClient** class provides access to a user's new email messages on a mail server. Methods are provided for listing available messages and then retrieving those messages, storing them either in files or in memory. Once a user's messages have been downloaded to the local system, they are typically removed from the server. This class is typically used in conjunction with the SocketTools.MailMessage class which is used to process the messages that are retrieved from the server. This class supports secure connections using TLS 1.2 and authentication using OAuth 2.0 bearer tokens.
RshClient	The **RshClient** class is used to execute a command on a server and return the output of that command to the client. This is most commonly used with UNIX based servers, although there are implementations of remote command servers for the Windows operating system. The class supports both the **rcmd** and **rshell** remote execution protocols. This class should not be used when connecting to a server over the Internet because the user credentials are not encrypted. For secure remote command execution and interactive terminal sessions, it is recommended you use the SocketTools.SshClient class.
SmtpClient	The **SmtpClient** class enables applications to deliver email messages to one or more recipients. The class provides an interface for addressing and delivering messages, and extended features such as user authentication and delivery status notification. There is no requirement to have certain third-party email applications installed or specific types of servers installed on the local system. This class supports secure connections using TLS 1.2 and authentication using OAuth 2.0 bearer tokens.
SocketWrench	The **SocketWrench** class is a general purpose networking class used to develop Internet and intranet applications using the TCP/IP protocol. With SocketWrench, you can create both client and server applications, as well as send and receive UDP datagrams. SocketWrench also supports secure connections using the Transport Layer Security (TLS) protocols. Enabling the security features of the class is done by setting a single property and does not require another .NET class to implement the encryption.
SshClient	The **SshClient** class is used to establish a secure connection with a server which provides a virtual terminal session for a user. Its functionality is similar to how character based consoles and serial terminals work, enabling a user to login to the server, execute commands and interact with applications running on the server. It also includes methods that enable a program to easily scan the data stream for specific sequences of characters, making it easy to create light-weight client interfaces to applications running on the server. File transfers using SFTP are implemented using the SocketTools.FtpClient class.
TelnetClient	The **TelnetClient** class is used to establish a connection with a server which provides a virtual terminal session for a user. Its functionality is similar to how character based consoles and serial terminals work,

	enabling a user to login to the server, execute commands and interact with applications running on the server. This class supports secure connections using TLS 1.2.
Terminal	The **Terminal** class is a Windows Forms control which provides a comprehensive interface for emulating an ANSI or DEC-VT220 terminal, with full support for all standard escape and control sequences, color mapping and other advanced features. The class provides both a high level interface for parsing escape sequences and updating a display, as well as lower level primitives for directly managing the virtual display, such as controlling the individual display cells, moving the cursor position and specifying display attributes.
TextMessage	The **TextMessage** class enables your application to send Short Message Service (SMS) messages using a service provider gateway. It submits messages to a wireless device on their network using standard email protocols. The clas provides methods that can be used to determine the provider associated with a specific telephone number and send a text message to the device using the provider's mail gateway. It does not require a third-party service to submit text messages, however it cannot be used to receive text messages.
WebLocation	The **WebLocation** class returns information about the location associated with the with the external IP address of the local system. The accuracy of this information can vary depending on the location, with the most detailed information being available for North America. The country and time zone information for all locations is generally accurate. However, as the location information becomes more precise, details such as city names, postal codes and specific geographic locations (e.g.: longitude and latitude) may have reduced accuracy.
WebStorage	The **WebStorage** class enables an application to securely store and manage private application data on a server. This class uses SocketTools Web Services and will only function if there is an active Internet connection and the local system is capable of establishing a secure connection to our servers.
WhoisClient	The **WhoisClient** class provides an interface for requesting registration information for an Internet domain name. When a domain name is registered, the organization that registers the domain must provide certain contact information along with technical information such as the primary name servers for that domain. The class provides an interface for requesting that information and returning it to the program so that it can be displayed or processed.

DnsClient Class

Implements the Domain Name Services protocol.

For a list of all members of this type, see DnsClient Members.

System.Object
  SocketTools.DnsClient

[Visual Basic]

```
Public Class DnsClient
    Implements IDisposable
```

[C#]

```
public class DnsClient : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The DnsClient class implements the Domain Name Services (DNS) protocol, which is used to resolve domain names into Internet addresses as well as provide other information about a domain. All of the SocketTools .NET classes provide basic domain name resolution functionality, but the DnsClient class gives an application direct control over what servers are queried, the amount of time spent waiting for a response and the type of information that is returned.

Requirements

Namespace: SocketTools

Assembly: SocketTools.DnsClient (in SocketTools.DnsClient.dll)

See Also

DnsClient Members | SocketTools Namespace

DnsClient Members

DnsClient overview

Public Static (Shared) Fields

dnsPortDefault	A constant value which specifies the default port number.
dnsRetries	A constant value which specifies the default number of retries.
dnsTimeout	A constant value which specifies the default timeout period.

Public Instance Constructors

DnsClient Constructor	Initializes a new instance of the DnsClient class.

Public Instance Fields

HostAlias	Returns the aliases for a given host name.
MailExchange	Returns the mail exchange servers for a given domain.
NameServer	Change or return the Internet address for a nameserver

Public Instance Properties

Handle	Returns the internal handle that is used by the class library.
HostAddress	Set or return the Internet address for the remote host.
HostAliases	Return the number of aliases for the specified host name.
HostFile	Set or return the name of an alternate host file.
HostInfo	Returns information about the host operating system.
HostName	Set or return the name of the remote host.
HostProtocol	Set the protocol to return service information for the specified host.
HostServices	Return the well-known services available for the specified host.
IsInitialized	Determine if the component has been initialized.
LastError	Set or return the last error that occurred.
LastErrorString	Return a description of the last error that occurred.
LocalAddress	Return the Internet address for the local system.
LocalDomain	Set or return the domain name for the local

	system.
LocalName	Return the host name for the local system.
MailExchanges	Return the number of mail exchange records for the specified host.
Options	Gets and sets a value which specifies one or more client options.
RemotePort	Set or return the port number used to establish a connection.
RemoteService	Set or return the name of the service associated with the remote port.
Retry	Set the number of times the control attempts to resolve a hostname.
ServerAddress	Return the address of the nameserver that resolved the query.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Enable or disable exceptions being raised when a method fails.
Timeout	Set or return the amount of time until a blocking operation fails.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Specify the socket function trace output file.
TraceFlags	Set or return the network function tracing flags.
Version	Gets a value which returns the current version of the DnsClient class library.

Public Instance Methods

AttachThread	Attach an instance of the class to the current thread
Cancel	Cancels the current blocking network operation.
Dispose	Overloaded. Releases all resources used by DnsClient.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initializes the component with the specified runtime license key.
MatchHost	Overloaded. Match a host name against one more

	strings that may contain wildcards.
Query	Perform a general nameserver query for a specific type of record.
Reset	Reset the internal state of the component.
Resolve	Resolve a hostname to an Internet address.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the component and unload the networking library.

Public Instance Events

OnCancel	Occurs when a blocking network operation is canceled using the Cancel method.
OnError	Occurs when a method fails.
OnTimeout	Occurs when a blocking network operation exceeds the timeout period specified by the Timeout property.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the DnsClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the client session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

DnsClient Class | SocketTools Namespace

DnsClient Constructor

Initializes a new instance of the DnsClient class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public DnsClient();
```

Example

The following example demonstrates creating an instance of the **DnsClient** class object and resolving a hostname into an Internet address using the **Resolve** method.

```
Dim dnsClient As SocketTools.DnsClient
Dim strHostName As String
Dim strHostAddress As String

dnsClient = New SocketTools.DnsClient
strHostName = TextBox1.Text.Trim()

If dnsClient.Resolve(strHostName, strHostAddress) Then
    StatusBar1.Text = "The Internet address for " + strHostName + " is " +
strHostAddress
Else
    StatusBar1.Text = "The Internet address for " + strHostName + " could not be
resolved"
End If
```

See Also

DnsClient Class | SocketTools Namespace

DnsClient Fields

The fields of the **DnsClient** class are listed below. For a complete list of **DnsClient** class members, see the DnsClient Members topic.

Public Static (Shared) Fields

dnsPortDefault	A constant value which specifies the default port number.
dnsRetries	A constant value which specifies the default number of retries.
dnsTimeout	A constant value which specifies the default timeout period.

Public Instance Fields

HostAlias	Returns the aliases for a given host name.
MailExchange	Returns the mail exchange servers for a given domain.
NameServer	Change or return the Internet address for a nameserver

See Also

DnsClient Class | SocketTools Namespace

DnsClient.HostAlias Field

Returns the aliases for a given host name.

[Visual Basic]
```
Public ReadOnly HostAlias As HostAliasArray
```

[C#]
```
public readonly HostAliasArray HostAlias;
```

Remarks

The **HostAlias** array returns the aliases assigned to the host specified by the **HostAddress** or **HostName** properties. If the host address or name can be resolved, the first element in the **HostAlias** array always refers to the host's fully qualified domain name.

The end of the alias list is indicated when the property returns an empty string. The array is zero based, meaning that the first index value is zero.

Example

```
Dim nIndex As Integer

ListBox1.Items.Clear()
dnsClient.HostName = strHostName

For nIndex = 0 To dnsClient.HostAliases - 1
    ListBox1.Items.Add(dnsClient.HostAlias(nIndex))
Next
```

See Also

DnsClient Class | SocketTools Namespace | HostAliasArray Class | HostAliases Property

DnsClient.MailExchange Field

Returns the mail exchange servers for a given domain.

[Visual Basic]

```
Public ReadOnly MailExchange As MailExchangeArray
```

[C#]

```
public readonly MailExchangeArray MailExchange;
```

Remarks

The **MailExchange** array returns the host name of the systems designated as the mail exchanges for the current domain. The mail exchange hosts are returned sorted in priority order, with the higher priority mail servers being listed first. The array is zero based, which means that the first index value is zero. The **HostName** property must be set to the domain name that you want to obtain the mail exchange records for.

This array is commonly used to determine which system is responsible for forwarding mail within a domain. For example, if a mail message is addressed to the user `someone@example.com`, you can determine the name of the server or servers responsible for accepting mail for that user by setting the value of the HostName property to `example.com` and then checking the **MailExchange** array. Note that it is possible that a domain will not have any mail exchange (MX) records, in which case you should attempt to to connect directly to a mail server running on the host specified in the domain name portion of the address.

Example

The following example populates a ListBox control with the host names of those servers responsible for accepting email for the specified domain:

```
Dim nIndex As Integer

ListBox1.Items.Clear()
dnsClient.HostName = strHostName

For nIndex = 0 To dnsClient.MailExchanges - 1
    ListBox1.Items.Add(dnsClient.MailExchange(nIndex))
Next
```

See Also

DnsClient Class | SocketTools Namespace | MailExchangeArray Class | MailExchanges Property

DnsClient.NameServer Field

Change or return the Internet address for a nameserver

[Visual Basic]

```
Public ReadOnly NameServer As NameServerArray
```

[C#]

```
public readonly NameServerArray NameServer;
```

Remarks

The **NameServer** array is used to specify one or more nameservers used to resolve hostnames and addresses. The address value must be an IP address in dot notation.

The index specifies which nameserver to set or return a value for. There may be up to four nameservers defined for any single instance of the component.

See Also

DnsClient Class | SocketTools Namespace

DnsClient Properties

The properties of the **DnsClient** class are listed below. For a complete list of **DnsClient** class members, see the DnsClient Members topic.

Public Instance Properties

Handle	Returns the internal handle that is used by the class library.
HostAddress	Set or return the Internet address for the remote host.
HostAliases	Return the number of aliases for the specified host name.
HostFile	Set or return the name of an alternate host file.
HostInfo	Returns information about the host operating system.
HostName	Set or return the name of the remote host.
HostProtocol	Set the protocol to return service information for the specified host.
HostServices	Return the well-known services available for the specified host.
IsInitialized	Determine if the component has been initialized.
LastError	Set or return the last error that occurred.
LastErrorString	Return a description of the last error that occurred.
LocalAddress	Return the Internet address for the local system.
LocalDomain	Set or return the domain name for the local system.
LocalName	Return the host name for the local system.
MailExchanges	Return the number of mail exchange records for the specified host.
Options	Gets and sets a value which specifies one or more client options.
RemotePort	Set or return the port number used to establish a connection.
RemoteService	Set or return the name of the service associated with the remote port.
Retry	Set the number of times the control attempts to resolve a hostname.
ServerAddress	Return the address of the nameserver that resolved the query.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.

ThrowError	Enable or disable exceptions being raised when a method fails.
Timeout	Set or return the amount of time until a blocking operation fails.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Specify the socket function trace output file.
TraceFlags	Set or return the network function tracing flags.
Version	Gets a value which returns the current version of the DnsClient class library.

See Also

DnsClient Class | SocketTools Namespace

DnsClient.Handle Property

Returns the internal handle that is used by the class library.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer value that specifies the client session handle. A return value of -1 indicates that no handle has been allocated for the class library.

See Also

DnsClient Class | SocketTools Namespace

DnsClient.HostAddress Property

Set or return the Internet address for the remote host.

[Visual Basic]
```
Public Property HostAddress As String
```

[C#]
```
public string HostAddress {get; set;}
```

Property Value

A string value which specifies the Internet address in dot notation.

Remarks

Setting the **HostAddress** property causes the control to submit a reverse query to the nameservers that you have specified. If a reverse entry is found for the IP address, the **HostName** property is changed to that host name.

Note that reverse domain name records (PTR records) may not be defined for a system and the application must not depend on this information being available.

See Also

DnsClient Class | SocketTools Namespace | HostName Property

DnsClient.HostAliases Property

Return the number of aliases for the specified host name.

[Visual Basic]

```
Public ReadOnly Property HostAliases As Integer
```

[C#]

```
public int HostAliases {get;}
```

Property Value

An integer value which specifies the number of host aliases.

Remarks

The **HostAliases** property returns the number of aliases for the host specified by the HostName property. If the specified host name cannot be resolved, this property will return a value of zero.

This property is typically used in conjunction with the **HostAlias** array.

Example

```
Dim nIndex As Integer

ListBox1.Items.Clear()
dnsClient.HostName = strHostName

For nIndex = 0 To dnsClient.HostAliases - 1
    ListBox1.Items.Add(dnsClient.HostAlias(nIndex))
Next
```

See Also

DnsClient Class | SocketTools Namespace | HostAlias Field

DnsClient.HostFile Property

Set or return the name of an alternate host file.

[Visual Basic]

```
Public Property HostFile As String
```

[C#]

```
public string HostFile {get; set;}
```

Property Value

A string which specifies a file name.

Remarks

The **HostFile** property is used to specify the name of an alternate file for resolving hostnames and IP addresses. The host file is used as a database that maps an IP address to one or more hostnames, and is used when setting the **HostName** or **HostAddress** properties and establishing a connection with a remote host. The file is a plain text file, with each line in the file specifying a record, and each field separated by spaces or tabs. The format of the file must be as follows:

```
address hostname [hostalias ...]
```

For example, one typical entry maps the name "localhost" to the local loopback IP address. This would be entered as:

```
127.0.0.1 localhost
```

The hash character (#) may be used to specify a comment in the file, and all characters after it are ignored up to the end of the line. Blank lines are ignored, as are any lines which do not follow the required format.

Setting this property loads the file into memory allocated for the current thread. If the contents of the file have changed after the method has been called, those changes will not be reflected when resolving hostnames or addresses. To reload the host file from disk, set the property again with the same file name. To remove the alternate host file from memory, specify an empty string as the file name.

If a host file has been specified, it is processed before the default host file when resolving a hostname into an IP address, or an IP address into a hostname. If the host name or address is not found, or no host file has been specified, a nameserver lookup is performed.

See Also

DnsClient Class | SocketTools Namespace

DnsClient.HostInfo Property

Returns information about the host operating system.

[Visual Basic]

```
Public ReadOnly Property HostInfo As String
```

[C#]

```
public string HostInfo {get;}
```

Property Value

A string which specifies the host operating system.

Remarks

The **HostInfo** property returns a string that includes the machine type and operating system for the host. This information corresponds to it's HINFO record in the database.

See Also

DnsClient Class | SocketTools Namespace

DnsClient.HostName Property

Set or return the name of the remote host.

[Visual Basic]

```
Public Property HostName As String
```

[C#]

```
public string HostName {get; set;}
```

Property Value

A string which specifies the host name.

Remarks

The **HostName** property should be set to the name of the remote system that you wish to obtain information on. Setting this property causes a query to be submitted to the nameservers that you have specified. If the host name can be resolved, the **HostAddress** property is set to the IP address of the host, in dot-notation.

See Also

DnsClient Class | SocketTools Namespace

DnsClient.HostProtocol Property

Set the protocol to return service information for the specified host.

[Visual Basic]

```
Public Property HostProtocol As ProtocolType
```

[C#]

```
public DnsClient.ProtocolType HostProtocol {get; set;}
```

Property Value

A ProtocolType enumeration type.

Remarks

The **HostProtocol** property determines the protocol for which service information is returned.

See Also

DnsClient Class | SocketTools Namespace | ProtocolType Enumeration

DnsClient.HostServices Property

Return the well-known services available for the specified host.

[Visual Basic]
```
Public ReadOnly Property HostServices As String
```

[C#]
```
public string HostServices {get;}
```

Property Value

A string that specifies the available services.

Remarks

The **HostServices** property returns a string that contains a list of well-known services for the specified host. This corresponds to the WKS entry in the nameserver's database. The services returned depend on the protocol specified in the **HostProtocol** property.

See Also

DnsClient Class | SocketTools Namespace

DnsClient.IsInitialized Property

Determine if the component has been initialized.

[Visual Basic]

```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]

```
public bool IsInitialized {get;}
```

Property Value

A boolean value which specifies if the component has been initialized.

Remarks

The **IsInitialized** property will return true if the object has been successfully initialized by the caller.

When an instance of the object is created, the class constructor will attempt to initialize itself using the runtime license key specified using the **RuntimeLicense** attribute. If no runtime license key has been specified, then the caller must explicitly initialize the object using the **Initialize** method.

See Also

DnsClient Class | SocketTools Namespace | Initialize Method | RuntimeLicense Attribute

DnsClient.LastError Property

Set or return the last error that occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public DnsClient.ErrorCode LastError {get; set;}
```

Property Value

An ErrorCode enumeration type.

Remarks

The **LastError** property can be read to determine the last error that occurred for this control. If a value is assigned to this property, it must either be zero to clear the error or a valid error code for the control.

See Also

DnsClient Class | SocketTools Namespace | ErrorCode Enumeration | LastErrorString Property

DnsClient.LastErrorString Property

Return a description of the last error that occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error.

Remarks

The **LastErrorString** property returns a description of the last error that occurred. This can be used to display a meaningful error message to a user, rather than just the numeric value returned by the **LastError** property.

See Also

DnsClient Class | SocketTools Namespace | LastError Property

DnsClient.LocalAddress Property

Return the Internet address for the local system.

[Visual Basic]
```
Public ReadOnly Property LocalAddress As String
```

[C#]
```
public string LocalAddress {get;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **LocalAddress** property returns the local system's Internet address in dot notation.

See Also

DnsClient Class | SocketTools Namespace | HostAddress Property | LocalName Property

DnsClient.LocalDomain Property

Set or return the domain name for the local system.

[Visual Basic]
```
Public Property LocalDomain As String
```

[C#]
```
public string LocalDomain {get; set;}
```

Property Value

A string that specifies the local domain name.

Remarks

The **LocalDomain** property is used to set the domain name for the local host. The local domain name is used when the name assigned to **HostName** property does not specify a domain (in other words, does not have a dot in the name). In that case, the value of the **LocalDomain** property is appended to the hostname.

If a domain name has been specified for the local system, the **LocalDomain** property is set to that value by default.

See Also

DnsClient Class | SocketTools Namespace | HostAddress Property | HostName Property

DnsClient.LocalName Property

Return the host name for the local system.

[Visual Basic]
```
Public ReadOnly Property LocalName As String
```

[C#]
```
public string LocalName {get;}
```

Property Value

A string which specifies the local host name.

Remarks

The **LocalName** property returns the name of the local host. The name that is returned depends on the configuration of the TCP/IP software.

See Also

DnsClient Class | SocketTools Namespace | HostName Property | LocalAddress Property

DnsClient.MailExchanges Property

Return the number of mail exchange records for the specified host.

[Visual Basic]

```
Public ReadOnly Property MailExchanges As Integer
```

[C#]

```
public int MailExchanges {get;}
```

Property Value

An integer value. A value of zero specifies that there are no mail exchange records for the host.

Remarks

The **MailExchanges** property returns the number of mail exchange (MX) records for the current host specified by the **HostName** property. This property can be used in conjunction with the **MailExchange** array to enumerate the servers responsible for accepting mail for a given domain.

Example

The following example populates a ListBox control with the host names of those servers responsible for accepting email for the specified domain:

```
Dim nIndex As Integer

ListBox1.Items.Clear()
dnsClient.HostName = strHostName

For nIndex = 0 To dnsClient.MailExchanges - 1
    ListBox1.Items.Add(dnsClient.MailExchange(nIndex))
Next
```

See Also

DnsClient Class | SocketTools Namespace | MailExchange Field

DnsClient.RemotePort Property

Set or return the port number used to establish a connection.

[Visual Basic]

```
Public Property RemotePort As Integer
```

[C#]

```
public int RemotePort {get; set;}
```

Property Value

An integer value which specifies the remote port number.

Remarks

The **RemotePort** property is used to set the port number that the control will use to establish a connection with the remote server. If the port number specifies a well-known port, the **RemoteService** property will be updated with that name.

The valid range for port numbers is 1 through 65535, inclusive. Specifying a port number outside of this range can cause an exception to be thrown. In most cases, it is recommended that you specify the default port number for the protocol.

See Also

DnsClient Class | SocketTools Namespace | RemoteService Property

DnsClient.Retry Property

Set the number of times the control attempts to resolve a hostname.

[Visual Basic]

```
Public Property Retry As Integer
```

[C#]

```
public int Retry {get; set;}
```

Property Value

An integer value.

Remarks

The **Retry** property specifies the number of times, per nameserver, that the control attempts to resolve a hostname or address. If attempts to query a nameserver fail, the control waits a period of time and then resubmits the query. As the number of retries increase, the longer the period of time the control waits to receive a response before attempting the query using another nameserver.

The default number of retries is four, with a minimum value of one and a maximum value of eight.

See Also

DnsClient Class | SocketTools Namespace | Timeout Property

DnsClient.ServerAddress Property

Return the address of the nameserver that resolved the query.

[Visual Basic]

```
Public ReadOnly Property ServerAddress As String
```

[C#]

```
public string ServerAddress {get;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **ServerAddress** property returns the Internet address of the nameserver that resolved the previous query.

See Also

DnsClient Class | SocketTools Namespace

DnsClient.ThreadModel Property

Gets and sets a value which specifies the threading model for the class instance.

[Visual Basic]

```
Public Property ThreadModel As ThreadingModel
```

[C#]

```
public DnsClient.ThreadingModel ThreadModel {get; set;}
```

Property Value

Returns one or more ThreadingModel enumeration value which specifies the threading model for the client. The default value for this property is **modelSingleThread**.

Remarks

The **ThreadModel** property specifies the threading model that is used by the class instance when a connection is established. The default value for this property is **modelSingleThread**, which specifies that only the thread that established the connection should be permitted to invoke methods. It is important to note that this threading model does not limit the application to a single thread of execution. When a session is established using the **Connect** method, that session is attached to the thread that created it. From that point on, until the session is terminated, only the owner may invoke methods in that instance of the class. The ownership of the class instance may be transferred from one thread to another using the **AttachThread** method.

Setting this property to **modelFreeThread** disables certain internal safety checks that are performed by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads. The application must ensure that no two threads will attempt to invoke a blocking method at the same time. In other words, if one thread invokes a method, the application must ensure that another thread will not attempt to invoke any other method at the same time using the same instance of the class.

See Also

DnsClient Class | SocketTools Namespace | AttachThread Method | ThreadingModel Enumeration | ThreadModel Attribute

DnsClient.ThrowError Property

Enable or disable exceptions being raised when a method fails.

[Visual Basic]

```
Public Property ThrowError As Boolean
```

[C#]

```
public bool ThrowError {get; set;}
```

Remarks

Error handling for method calls can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to false, the application should check the return value of any method that is used, and report errors based upon the documented value of the **LastError** property. It is the responsibility of the application to interpret the error code and act accordingly.

If the **ThrowError** property is set to true, then errors that occur when calling a method will cause an exception to be raised. The application must handle the exception using the error handling facilities in the language being used. For example, in Visual Basic, the Try..Catch..End Try statements can be used.

Note that if an error occurs while a property value is being accessed, an exception will always be raised regardless of the value of this property.

Example

The following examples demonstrate the difference between handling errors when calling the **Resolve** method. In the first example, the **ThrowError** property is set to False, which means that the **Resolve** method will return True or False, depending on whether it succeeds or fails. If it returns False, the **LastError** and **LastErrorString** properties can be used to determine the error.

In the second example, the **ThrowError** property is set to True, which causes the **Resolve** method to throw an exception if an error occurs. This is handled by the Try..Catch..End Try block, and the **DnsClientException** class provides information about the error.

```
dnsClient.ThrowError = False

If dnsClient.Resolve(strHostName, strHostAddress) = False Then
    MsgBox(dnsClient.LastErrorString, MsgBoxStyle.Exclamation)
    Exit Sub
End If


dnsClient.ThrowError = True

Try
    dnsClient.Resolve(strHostName, strHostAddress)
Catch ex As SocketTools.DnsClientException
    MsgBox(ex.Message, MsgBoxStyle.Exclamation)
End Try
```

See Also

DnsClient Class | SocketTools Namespace | DnsClientException Class | LastError Property | LastErrorString Property

DnsClient.Timeout Property

Set or return the amount of time until a blocking operation fails.

[Visual Basic]
```
Public Property Timeout As Integer
```

[C#]
```
public int Timeout {get; set;}
```

Property Value

An integer value which specifies the timeout period in seconds.

Remarks

Setting the **Timeout** property specifies the number of seconds until a blocking operation fails and the method returns to the caller.

Note that the **Timeout** property also determines the amount of time the component will spend attempting to connect to a remote host. If a connection is not established within the given time period, the connection attempt will fail.

See Also

DnsClient Class | SocketTools Namespace | Retry Property

DnsClient.Trace Property

Gets and sets a value which indicates if network function logging is enabled.

[Visual Basic]

```
Public Property Trace As Boolean
```

[C#]

```
public bool Trace {get; set;}
```

Property Value

A boolean value.

Remarks

The **Trace** property is used to enable or disable the logging of Windows Sockets function calls. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the networking components, and is enabled or disabled for an entire process. This means that once tracing is enabled for a given object, all of the function calls made by the process using any of the SocketTools components will be logged. For example, if you have an application using both the FTP and POP3 components, and you set the **Trace** property to True on the FTP component, function calls made by both the FTP and POP3 components will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all components used by the process.

If tracing is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the trace file is fairly large. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

Note that only those function calls made by the SocketTools networking components will be logged. Calls made directly to the Windows Sockets API, or calls made by other components, will not be logged.

See Also

DnsClient Class | SocketTools Namespace | TraceFile Property | TraceFlags Property

DnsClient.TraceFile Property

Specify the socket function trace output file.

[Visual Basic]
```
Public Property TraceFile As String
```

[C#]
```
public string TraceFile {get; set;}
```

Property Value

A string which specifies a file name.

Remarks

TraceFile property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named `SocketTools.log` is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the controls being used. If multiple controls have tracing enabled, the **TraceFile** property should be set to the same value for each control. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace logfile has the following format:

```
    TestApp 105020 0000 INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27)
returned 0
TestApp 105020 0015 WRN: connect(46, 192.0.0.1:1234, 16) returned -1
[10035]
TestApp 111535 0000 ERR: accept(46, NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column is the local time in hours, minutes and seconds. The third column is the elapsed time in milliseconds since the previous function call. The fourth column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the control is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

If the specified logfile cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

DnsClient Class | SocketTools Namespace | Trace Property | TraceFlags Property

DnsClient.TraceFlags Property

Set or return the network function tracing flags.

[Visual Basic]
```
Public Property TraceFlags As TraceOptions
```

[C#]
```
public DnsClient.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration type.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Since function logging is enabled per-process, the trace flags are shared by all instances of the components being used. If multiple components have tracing enabled, the **TraceFlags** property should be set to the same value for each component. Changing the trace flags for any one instance of the component will affect the logging performed for all components used by the application.

Warnings are generated when a non-fatal error is returned by a Windows Sockets function. For example, if data is being written through the control and an error indicating that the operation would block is returned, only a warning is logged since the application simply needs to attempt to write the data at a later time

See Also

DnsClient Class | SocketTools Namespace | Trace Property | TraceFile Property | TraceOptions Enumeration

DnsClient.Version Property

Gets a value which returns the current version of the DnsClient class library.

[Visual Basic]

```
Public ReadOnly Property Version As String
```

[C#]

```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the DnsClient class library. This value can be used by an application for validation and debugging purposes.

See Also

DnsClient Class | SocketTools Namespace

DnsClient Methods

The methods of the **DnsClient** class are listed below. For a complete list of **DnsClient** class members, see the DnsClient Members topic.

Public Instance Methods

AttachThread	Attach an instance of the class to the current thread
Cancel	Cancels the current blocking network operation.
Dispose	Overloaded. Releases all resources used by DnsClient.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initializes the component with the specified runtime license key.
MatchHost	Overloaded. Match a host name against one more strings that may contain wildcards.
Query	Perform a general nameserver query for a specific type of record.
Reset	Reset the internal state of the component.
Resolve	Resolve a hostname to an Internet address.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the component and unload the networking library.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the DnsClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the client session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

DnsClient Class | SocketTools Namespace

DnsClient.Cancel Method

Cancels the current blocking network operation.

[Visual Basic]

```
Public Function Cancel() As Boolean
```

[C#]

```
public bool Cancel();
```

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

When the **Cancel** method is called, the blocked method may not immediately return. An internal flag is set which causes the blocking operation to exit with an error. This means that the application cannot cancel an operation and immediately perform some other operation. It must allow the calling stack to unwind, returning control back to the blocking method before calling any other methods.

Note that if the **ThrowError** property is set to True, calling the **Cancel** method will cause the blocked method to raise an exception which must be handled by the application.

See Also

DnsClient Class | SocketTools Namespace

DnsClient.Dispose Method

Releases all resources used by DnsClient.

Overload List

Releases all resources used by DnsClient.

public void Dispose();

Releases the unmanaged resources allocated by the DnsClient class and optionally releases the managed resources.

protected virtual void Dispose(bool);

See Also

DnsClient Class | SocketTools Namespace

DnsClient.Dispose Method ()

Releases all resources used by DnsClient.

[Visual Basic]

```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]

```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

DnsClient Class | SocketTools Namespace | DnsClient.Dispose Overload List

DnsClient.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the DnsClient class and optionally releases the managed resources.

[Visual Basic]

```
Overridable Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]

```
protected virtual void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **DnsClient** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

DnsClient Class | SocketTools Namespace | DnsClient.Dispose Overload List

DnsClient.Finalize Method

Destroys an instance of the class, releasing the resources allocated for the client session and unloading the networking library.

[Visual Basic]

```
Overrides Protected Sub Finalize()
```

[C#]

```
protected override void Finalize();
```

See Also

DnsClient Class | SocketTools Namespace

DnsClient.Initialize Method

Initializes the component with the default runtime license key.

Overload List

Initializes the component with the default runtime license key.

public bool Initialize();

Initializes the component with the specified runtime license key.

public bool Initialize(string);

See Also

DnsClient Class | SocketTools Namespace | RuntimeLicenseAttribute Class | Uninitialize Method

DnsClient.Initialize Method ()

Initializes the component with the default runtime license key.

[Visual Basic]
```
Overloads Public Function Initialize() As Boolean
```

[C#]
```
public bool Initialize();
```

Return Value

A boolean value which specifies if the component was initialized successfully with the default runtime license key. A return value of False indicates that the license key is not valid.

Remarks

The Initialize method is used to explicitly initialize the component with the runtime license key that has been specified using the RuntimeLicense attribute. Normally it is not necessary to call this method because the component will automatically be initialized when the class constructor is called. This method should only be called if the **Uninitialize** method was previously called.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use SocketTools components.

See Also

DnsClient Class | SocketTools Namespace | DnsClient.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

DnsClient.Initialize Method (String)

Initializes the component with the specified runtime license key.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Parameters

licenseKey

A string which specifies the runtime license key.

Return Value

A boolean value which specifies if the component was initialized successfully with the runtime license key. A return value of False indicates that the license key is not valid.

Remarks

The Initialize method is used to explicitly initialize the component with a runtime license key, if one has not been specified using the RuntimeLicense attribute.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use SocketTools components.

See Also

DnsClient Class | SocketTools Namespace | DnsClient.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

DnsClient.MatchHost Method

Match a host name against one more strings that may contain wildcards.

Overload List

Match a host name against one more strings that may contain wildcards.

public bool MatchHost(string,string);

Match a host name against one more strings that may contain wildcards.

public bool MatchHost(string,string,bool);

See Also

DnsClient Class | SocketTools Namespace

DnsClient.MatchHost Method (String, String)

Match a host name against one more strings that may contain wildcards.

```
[Visual Basic]
Overloads Public Function MatchHost( _
   ByVal hostName As String, _
   ByVal hostMask As String _
) As Boolean
```

```
[C#]
public bool MatchHost(
   string hostName,
   string hostMask
);
```

Parameters

hostName

A string which specifies the host name or Internet address to match against the host mask string.

hostMask

A string which specifies one or more values to match against the host name. The asterisk character can be used to match any number of characters in the host name, and the question mark can be used to match any single character. Multiple values may be specified by separating them with a semicolon.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

The **MatchHost** method provides a convenient way for an application to determine if a given host name matches one or more mask strings which may contain wildcard characters. For example, the host name could be "www.microsoft.com" and the host mask string could be "*.microsoft.com". In this example, the function would return True, indicating the host name matched the mask. However, if the mask string was "*.net" then the function would return False, indicating that there was no match. Multiple mask values can be combined by separating them with a semicolon; for example, the mask "*.com;*.org" would match any host name in either the .com or .org top-level domains.

See Also

DnsClient Class | SocketTools Namespace | DnsClient.MatchHost Overload List

DnsClient.MatchHost Method (String, String, Boolean)

Match a host name against one more strings that may contain wildcards.

[Visual Basic]

```
Overloads Public Function MatchHost( _
   ByVal hostName As String, _
   ByVal hostMask As String, _
   ByVal resolve As Boolean _
) As Boolean
```

[C#]

```
public bool MatchHost(
   string hostName,
   string hostMask,
   bool resolve
);
```

Parameters

hostName
A string which specifies the host name or Internet address to match against the host mask string.

hostMask
A string which specifies one or more values to match against the host name. The asterisk character can be used to match any number of characters in the host name, and the question mark can be used to match any single character. Multiple values may be specified by separating them with a semicolon.

resolve
A boolean value which determines if the host name or address should be resolved when matching the host against the mask string. If this parameter is True, two checks against the host mask string will be performed; once for the host name specified and once for its IP address. If this parameter is False, then the match is made only against the host name string provided.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

The **MatchHost** method provides a convenient way for an application to determine if a given host name matches one or more mask strings which may contain wildcard characters. For example, the host name could be "www.microsoft.com" and the host mask string could be "*.microsoft.com". In this example, the function would return True, indicating the host name matched the mask. However, if the mask string was "*.net" then the function would return False, indicating that there was no match. Multiple mask values can be combined by separating them with a semicolon; for example, the mask "*.com;*.org" would match any host name in either the .com or .org top-level domains.

See Also

DnsClient Class | SocketTools Namespace | DnsClient.MatchHost Overload List

DnsClient.Query Method

Perform a general nameserver query for a specific type of record.

[Visual Basic]

```
Public Function Query( _
   ByVal hostName As String, _
   ByVal recordType As RecordType, _
   ByRef recordData As String _
) As Boolean
```

[C#]

```
public bool Query(
   string hostName,
   RecordType recordType,
   ref string recordData
);
```

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

The **Query** method performs a general nameserver query for a given record based on the name and type. This method will not use a local host file when performing host name or address queries.

To resolve a hostname into an Internet address, it is recommended that you use the **Resolve** method.

Example

The following example queries a nameserver for the TXT record for the specified remote host. If a record is found, its value will be displayed in a message box.

```
Dim strRecord As String

If dnsClient.Query(TextBox1.Text, RecordType.dnsRecordTXT, strRecord) Then
    MsgBox(strRecord, MsgBoxStyle.Information)
End If
```

See Also

DnsClient Class | SocketTools Namespace | HostAddress Property | HostName Property | Resolve Method

DnsClient.Reset Method

Reset the internal state of the component.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method resets the internal state of the component and restores property values to their defaults.

See Also

DnsClient Class | SocketTools Namespace

DnsClient.Resolve Method

Resolve a hostname to an Internet address.

[Visual Basic]

```
Public Function Resolve( _
   ByVal hostName As String, _
   ByRef hostAddress As String _
) As Boolean
```

[C#]

```
public bool Resolve(
   string hostName,
   ref string hostAddress
);
```

Parameters

hostName
A string which specifies the hostname to resolve.

hostAddress
A string which will contain the Internet address for the specified hostname when the method returns. If the hostname cannot be resolved, this parameter will be set to an empty string.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

The **Resolve** method is used to convert a host name into an Internet address. Note that unlike the **Query** method, this method will use the local host file when resolving the host name.

See Also

DnsClient Class | SocketTools Namespace

DnsClient.Uninitialize Method

Uninitialize the component and unload the networking library.

[Visual Basic]
```
Public Sub Uninitialize()
```

[C#]
```
public void Uninitialize();
```

Remarks

The Uninitialize method explicitly releases resources allocated by the component. Normally it is not required that the application call this method since it is automatically called by the class destructor. This method should only be used if the **Initialize** method was explicitly called.

See Also

DnsClient Class | SocketTools Namespace | Initialize Method

DnsClient Events

The events of the **DnsClient** class are listed below. For a complete list of **DnsClient** class members, see the DnsClient Members topic.

Public Instance Events

⚡ OnCancel	Occurs when a blocking network operation is canceled using the Cancel method.
⚡ OnError	Occurs when a method fails.
⚡ OnTimeout	Occurs when a blocking network operation exceeds the timeout period specified by the Timeout property.

See Also

DnsClient Class | SocketTools Namespace

DnsClient.OnCancel Event

Occurs when a blocking network operation is canceled using the Cancel method.

[Visual Basic]

```
Public Event OnCancel As EventHandler
```

[C#]

```
public event EventHandler OnCancel;
```

Remarks

The event handler receives an argument of type EventArgs.

See Also

DnsClient Class | SocketTools Namespace

DnsClient.OnError Event

Occurs when a method fails.

[Visual Basic]
```
Public Event OnError As OnErrorEventHandler
```

[C#]
```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type DnsClient.ErrorEventArgs containing data related to this event. The following **DnsClient.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Returns a description of the last error that occurred.
Error	Returns the value of the last error that has occurred.

Remarks

The event handler receives an argument of type ErrorEventArgs containing data related to this event. The following ErrorEventArgs properties provide information specific to this event.

Property	**Description**
Error	Returns the value of the last error that has occurred
Description	Returns a description of the last error that occurred

See Also

DnsClient Class | SocketTools Namespace

DnsClient.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see DnsClient.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.DnsClient.ErrorEventArgs

[Visual Basic]

```
Public Class DnsClient.ErrorEventArgs
    Inherits EventArgs
```

[C#]

```
public class DnsClient.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.DnsClient (in SocketTools.DnsClient.dll)

See Also

DnsClient.ErrorEventArgs Members | SocketTools Namespace

DnsClient.ErrorEventArgs Members

DnsClient.ErrorEventArgs overview

Public Instance Constructors

DnsClient.ErrorEventArgs Constructor	Initializes a new instance of the DnsClient.ErrorEventArgs class.

Public Instance Properties

Description	Returns a description of the last error that occurred.
Error	Returns the value of the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

DnsClient.ErrorEventArgs Class | SocketTools Namespace

DnsClient.ErrorEventArgs Constructor

Initializes a new instance of the DnsClient.ErrorEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public DnsClient.ErrorEventArgs();
```

See Also

DnsClient.ErrorEventArgs Class | SocketTools Namespace

DnsClient.ErrorEventArgs Properties

The properties of the **DnsClient.ErrorEventArgs** class are listed below. For a complete list of **DnsClient.ErrorEventArgs** class members, see the DnsClient.ErrorEventArgs Members topic.

Public Instance Properties

Description	Returns a description of the last error that occurred.
Error	Returns the value of the last error that has occurred.

See Also

DnsClient.ErrorEventArgs Class | SocketTools Namespace

DnsClient.ErrorEventArgs.Description Property

Returns a description of the last error that occurred.

[Visual Basic]
```
Public ReadOnly Property Description As String
```

[C#]
```
public string Description {get;}
```

Property Value

String which describes the error.

See Also

DnsClient.ErrorEventArgs Class | SocketTools Namespace

DnsClient.ErrorEventArgs.Error Property

Returns the value of the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public DnsClient.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

DnsClient.ErrorEventArgs Class | SocketTools Namespace

DnsClient.OnTimeout Event

Occurs when a blocking network operation exceeds the timeout period specified by the Timeout property.

[Visual Basic]

```
Public Event OnTimeout As EventHandler
```

[C#]

```
public event EventHandler OnTimeout;
```

Remarks

The event handler receives an argument of type EventArgs.

See Also

DnsClient Class | SocketTools Namespace

DnsClient.ErrorCode Enumeration

Specifies the error codes returned by the DnsClient class.

[Visual Basic]
```
Public Enum DnsClient.ErrorCode
```

[C#]
```
public enum DnsClient.ErrorCode
```

Remarks

The DnsClient class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.
errorProxyAuthenticationRequired	Proxy authentication required.

errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.
errorUnsupportedMediaType	Request specified an unsupported media type.

errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.
errorNoMoreMailboxes	No more mailboxes exist on this server.

errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.
errorInvalidThreadHandle	The specified thread handle is no longer valid.

errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been defined.

errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or unrecognized.

errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.

Requirements

Namespace: SocketTools

Assembly: SocketTools.DnsClient (in SocketTools.DnsClient.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

DnsClient.ProtocolType Enumeration

Provides constant values for the HostProtocol property.

[Visual Basic]

```
Public Enum DnsClient.ProtocolType
```

[C#]

```
public enum DnsClient.ProtocolType
```

Members

Member Name	Description
protocolTCP	Transmission control protocol.
protocolUDP	User datagram protocol.
protocolDefault	The default protocol; the same value as ProtocolType.protocolTCP.

Requirements

Namespace: SocketTools

Assembly: SocketTools.DnsClient (in SocketTools.DnsClient.dll)

See Also

SocketTools Namespace | HostProtocol Property

DnsClient.RecordType Enumeration

Provides constant values for record queries

[Visual Basic]
```
Public Enum DnsClient.RecordType
```

[C#]
```
public enum DnsClient.RecordType
```

Remarks

The **dnsRecordAddress** record type is used to resolve a host name into an Internet address, and is the most common type of nameserver query that is performed. This is the type of query that is used when you set the **HostName** property and then read the **HostAddress** property to obtain its address.

The **dnsRecordPTR** record type is used to resolve an Internet address into a host name, and is also referred to as a reverse DNS lookup. This is the same type of query that is performed when you set the **HostAddress** property and then read the **HostName** property to determine the host name.

Members

Member Name	Description
recordNone	No record type.
recordAddress	Host address record.
recordNS	Authoritative nameserver record.
recordCNAME	Canonical name (alias) record.
recordSOA	Start of Authority record.
recordWKS	Well known services record.
recordPTR	Domain name.
recordHINFO	Host information record.
recordMINFO	Mailbox information record.
recordMX	Mail exchange host record.
recordTXT	Text resource record.
recordRP	Responsible Person (RP) resource record.
recordX25	X.25 resource record.
recordISDN	ISDN resource record.
recordRT	Route Through (RT) resource record.
recordAAAA	IPv6 address record.
recordLOC	Location resource record.
recordUINFO	User information.
recordUID	User ID record.
recordGID	Group ID record.

Requirements

Namespace: SocketTools

Assembly: SocketTools.DnsClient (in SocketTools.DnsClient.dll)

See Also

SocketTools Namespace

DnsClient.TraceOptions Enumeration

Specifies the logging options that the DnsClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum DnsClient.TraceOptions
```

```
[C#]
[Flags]
public enum DnsClient.TraceOptions
```

Remarks

The DnsClient class uses the **TraceOptions** enumeration to specify what kind of debugging information is written to the trace logfile. These options are only meaningful when trace logging is enabled by setting the **Trace** property to **true**.

Members

Member Name	Description	Value
traceDefault	The default trace logging option. This is the same as specifying the **traceInfo** option.	0
traceInfo	All network function calls are written to the trace file. This is the default value.	0
traceError	Only those network function calls which fail are recorded in the trace file.	1
traceWarning	Only those network function calls which fail, or return values which indicate a warning, are recorded in the trace file.	2
traceHexDump	All network function calls are written to the trace file, plus all the data that is sent or received is displayed, in both ASCII and hexadecimal format.	4
traceProcess	All function calls in the current process are logged, rather than only those functions in the current thread. This option is useful for multithreaded applications that are using worker threads.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.DnsClient (in SocketTools.DnsClient.dll)

See Also

SocketTools Namespace

DnsClient.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub DnsClient.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void DnsClient.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender
The source of the event.

e
An ErrorEventArgs that contains the event data.

Remarks

The event handler receives an argument of type ErrorEventArgs containing data related to this event. The following ErrorEventArgs properties provide information specific to this event.

Property	Description
Error	Returns the value of the last error that has occurred
Description	Returns a description of the last error that occurred

Requirements

Namespace: SocketTools

Assembly: SocketTools.DnsClient (in SocketTools.DnsClient.dll)

See Also

SocketTools Namespace

DnsClient.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see DnsClient.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.DnsClient.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class DnsClient.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class DnsClient.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use SocketTools components.

Example

```
<Assembly: SocketTools.DnsClient.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.DnsClient.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.DnsClient (in SocketTools.DnsClient.dll)

See Also

DnsClient.RuntimeLicenseAttribute Members | SocketTools Namespace

DnsClient.RuntimeLicenseAttribute Members

DnsClient.RuntimeLicenseAttribute overview

Public Instance Constructors

DnsClient.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

DnsClient.RuntimeLicenseAttribute Class | SocketTools Namespace

DnsClient.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public DnsClient.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use SocketTools components.

See Also

DnsClient.RuntimeLicenseAttribute Class | SocketTools Namespace

DnsClient.RuntimeLicenseAttribute Properties

The properties of the **DnsClient.RuntimeLicenseAttribute** class are listed below. For a complete list of **DnsClient.RuntimeLicenseAttribute** class members, see the DnsClient.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

DnsClient.RuntimeLicenseAttribute Class | SocketTools Namespace

DnsClient.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

DnsClient.RuntimeLicenseAttribute Class | SocketTools Namespace

DnsClientException Class

This exception is thrown when a networking error occurs.

For a list of all members of this type, see DnsClientException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.DnsClientException

[Visual Basic]
```
Public Class DnsClientException
    Inherits ApplicationException
```

[C#]
```
public class DnsClientException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

This exception can be thrown when a property is set that results in a networking error, or as the result of calling a method that fails when the **ThrowError** property has been set to True.

Requirements

Namespace: SocketTools

Assembly: SocketTools.DnsClient (in SocketTools.DnsClient.dll)

See Also

DnsClientException Members | SocketTools Namespace

DnsClientException Members

DnsClientException overview

Public Instance Constructors

DnsClientException	Overloaded. Initializes a new instance of the DnsClientException class.

Public Instance Properties

HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Returns a message that describes the current exception.
Number	Return the last error that occurred.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

DnsClientException Class | SocketTools Namespace

DnsClientException Constructor

Initializes a new instance of the DnsClientException class without a message.

Overload List

Initializes a new instance of the DnsClientException class without a message.

public DnsClientException();

Initializes a new instance of the DnsClientException class with the specific error code value.

public DnsClientException(ErrorCode);

Initializes a new instance of the DnsClientException class with the specific error code value.

public DnsClientException(int);

Initializes a new instance of the DnsClientException class with a message.

public DnsClientException(string);

Initializes a new instance of the DnsClientException class with a message and a reference to the inner exception that is the cause of this exception.

public DnsClientException(string,Exception);

See Also

DnsClientException Class | SocketTools Namespace

DnsClientException Constructor ()

Initializes a new instance of the DnsClientException class without a message.

[Visual Basic]

```
Overloads Public Sub New()
```

[C#]

```
public DnsClientException();
```

See Also

DnsClientException Class | SocketTools Namespace | DnsClientException Constructor Overload List

DnsClientException Constructor (String)

Initializes a new instance of the DnsClientException class with a message.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]

```
public DnsClientException(
   string message
);
```

Parameters

message

The message to display with this exception.

See Also

DnsClientException Class | SocketTools Namespace | DnsClientException Constructor Overload List

DnsClientException Constructor (String, Exception)

Initializes a new instance of the DnsClientException class with a message and a reference to the inner exception that is the cause of this exception.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal inner As Exception _
)
```

[C#]

```
public DnsClientException(
   string message,
   Exception inner
);
```

Parameters

message

The message to display with this exception.

inner

The exception that is the cause of the current exception. If the inner parameter is not a null reference (Nothing in Visual Basic), the current exception is raised in a catch block that handles the inner exception.

See Also

DnsClientException Class | SocketTools Namespace | DnsClientException Constructor Overload List

DnsClientException Constructor (Int32)

Initializes a new instance of the DnsClientException class with the specific error code value.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

[C#]

```
public DnsClientException(
   int code
);
```

Parameters

code

An integer value which specifies the error that caused the exception to be raised.

See Also

DnsClientException Class | SocketTools Namespace | DnsClientException Constructor Overload List

DnsClientException Constructor (ErrorCode)

Initializes a new instance of the DnsClientException class with the specific error code value.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal code As ErrorCode _
)
```

[C#]

```
public DnsClientException(
   ErrorCode code
);
```

Parameters

code

An ErrorCode enumeration which specifies the error that caused the exception to be raised.

See Also

DnsClientException Class | SocketTools Namespace | DnsClientException Constructor Overload List

DnsClientException Properties

The properties of the **DnsClientException** class are listed below. For a complete list of **DnsClientException** class members, see the DnsClientException Members topic.

Public Instance Properties

HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Returns a message that describes the current exception.
Number	Return the last error that occurred.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

DnsClientException Class | SocketTools Namespace

DnsClientException.Message Property

Returns a message that describes the current exception.

[Visual Basic]
```
Overrides Public ReadOnly Property Message As String
```

[C#]
```
public override string Message {get;}
```

Property Value

A string that explains the reason for the exception, or an empty string.

Remarks

The **Message** property returns a description of the last error that caused the exception. This can be used to display a meaningful error message to a user, rather than just the numeric value returned by the **Number** property.

See Also

DnsClientException Class | SocketTools Namespace

DnsClientException.Number Property

Return the last error that occurred.

[Visual Basic]
```
Public ReadOnly Property Number As Integer
```

[C#]
```
public int Number {get;}
```

Property Value

An ErrorCode enumeration type that specifies the error that caused the exception.

See Also

DnsClientException Class | SocketTools Namespace

DnsClientException Methods

The methods of the **DnsClientException** class are listed below. For a complete list of **DnsClientException** class members, see the DnsClientException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

DnsClientException Class | SocketTools Namespace

DnsClientException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]
```
Overrides Public Function ToString() As String
```

[C#]
```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

ToString returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by ToString is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

The default implementation of ToString obtains the description of the error that caused the exception. If there is no error message or if it is an empty string (""), then no error message is returned.

This method overrides Object.ToString.

See Also

DnsClientException Class | SocketTools Namespace

FileEncoder Class

Encode and decode files using standard algorithms such as base64, uuencode and quoted-printable. The class can also be used to compress and expand files, either separately or as part of the encoding process.

For a list of all members of this type, see FileEncoder Members.

System.Object
  SocketTools.FileEncoder

[Visual Basic]
```
Public Class FileEncoder
    Implements IDisposable
```

[C#]
```
public class FileEncoder : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The FileEncoding class provides methods for encoding and decoding binary files, typically attachments to email messages. The process of encoding converts the contents of a binary file to printable 7-bit ASCII text. Decoding reverses the process, converting a previously encoded text file back into a binary file.

There are two primary types of encoding used, uucode and base64. The uucode algorithm (so called because the programs to perform the conversion were called uuencode and uudecode) is commonly used on UNIX systems and is still widely used when attaching binary files to USENET newsgroup posts. The base64 algorithm is most commonly used with email attachments, and is often referred to as MIME encoding since this is the encoding method specified in the MIME standards document. The class also supports a newer encoding format called yEnc, which has become popular for posting binary files to Usenet newsgroups.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileEncoder (in SocketTools.FileEncoder.dll)

See Also

FileEncoder Members | SocketTools Namespace

FileEncoder Members

FileEncoder overview

Public Instance Constructors

FileEncoder Constructor	Initializes a new instance of the FileEncoder class.

Public Instance Properties

DecodedText	Set the value of the current decoded text string, or return the decoded value of an encoded string.
DecryptedText	Gets and sets the value of the current decrypted string.
EncodedText	Set the value of the current encoded text string, or return the encoded value of a plain text string.
Encoding	Get or set the current encoding method.
EncryptedText	Gets and sets the value of the current encrypted string.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
Password	Gets and sets the password used to encrypt and decrypt data.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Version	Gets a value which returns the current version of the FileEncoder class library.

Public Instance Methods

CompressData	Compresses the contents of the specified buffer.
CompressFile	Overloaded. Compress the contents of the specified file.
DecodeFile	Overloaded. Decodes an encoded file, storing the contents in the specified file.
DecryptData	Overloaded. Decrypts the contents of a byte array.
DecryptFile	Overloaded. Decrypts the contents of a file using AES-256 encryption.
Dispose	Overloaded. Releases all resources used by FileEncoder.

EncodeFile	Overloaded. Encodes a file, storing the contents as printable text in the specified file.
EncryptData	Overloaded. Encrypts the contents of a byte array.
EncryptFile	Overloaded. Encrypts the contents of a file using AES-256 encryption.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
ExpandData	Expands the contents of the compressed buffer.
ExpandFile	Overloaded. Expands the contents of a previously compressed file.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initializes the component with the specified runtime license key.
Reset	Reset the internal state of the class back to its defaults.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the component.

Public Instance Events

OnError	Occurs when a method fails.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the FileEncoder class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the object.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FileEncoder Class | SocketTools Namespace

FileEncoder Constructor

Initializes a new instance of the FileEncoder class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FileEncoder();
```

See Also

FileEncoder Class | SocketTools Namespace

FileEncoder Properties

The properties of the **FileEncoder** class are listed below. For a complete list of **FileEncoder** class members, see the FileEncoder Members topic.

Public Instance Properties

DecodedText	Set the value of the current decoded text string, or return the decoded value of an encoded string.
DecryptedText	Gets and sets the value of the current decrypted string.
EncodedText	Set the value of the current encoded text string, or return the encoded value of a plain text string.
Encoding	Get or set the current encoding method.
EncryptedText	Gets and sets the value of the current encrypted string.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
Password	Gets and sets the password used to encrypt and decrypt data.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Version	Gets a value which returns the current version of the FileEncoder class library.

See Also

FileEncoder Class | SocketTools Namespace

FileEncoder.DecodedText Property

Set the value of the current decoded text string, or return the decoded value of an encoded string.

[Visual Basic]

```
Public Property DecodedText As String
```

[C#]

```
public string DecodedText {get; set;}
```

Property Value

Returns a string which contains the decoded value of an encoded string.

Remarks

The **DecodedText** property is used to specify the unencoded value of a string, or return the decoded value of a previously encoded string. If the property is set, the current plain (unencoded) text is changed to that value and reading the **EncodedText** property will return the encoded string for this value. If the property is read, it will return the decoded value of the **EncodedText** property.

The class will use the value of the **Encoding** property to determine how the text should be decoded. Note that only base64 and quoted-printable encoding is supported when decoding a string using this property. To decode the contents of a file, it is recommended that you use the **DecodeFile** method.

See Also

FileEncoder Class | SocketTools Namespace

FileEncoder.EncodedText Property

Set the value of the current encoded text string, or return the encoded value of a plain text string.

[Visual Basic]

```
Public Property EncodedText As String
```

[C#]

```
public string EncodedText {get; set;}
```

Property Value

Returns a string which contains the encoded value of a plain text string.

Remarks

The **EncodedText** property is used to specify the encoded value of a string, or return the encoded value of a plain text (decoded) string. If the property is set, the current value will be changed to the encoded value, and the **DecodedText** property will return the plain text value of the encoded string. If the property is read, it will return the encoded value of the **DecodedText** property.

The class will use the value of the **Encoding** property to determine how the text should be encoded. Note that only base64 and quoted-printable encoding is supported when encoding a string using this property. To encode the contents of a file, it is recommended that you use the **EncodeFile** method.

See Also

FileEncoder Class | SocketTools Namespace

FileEncoder.Encoding Property

Get or set the current encoding method.

[Visual Basic]
```
Public Property Encoding As EncodingType
```

[C#]
```
public FileEncoder.EncodingType Encoding {get; set;}
```

Property Value

Returns an EncodingType enumeration value which specifies the current encoding method.

Remarks

The **Encoding** property determines how the specified file will be encoded or decoded by the class. The value of this property specifies the default encoding method for the **DecodeFile** and **EncodeFile** methods. Unless needed for a specific purpose, it is strongly recommended that binary files be encoded with the base64 algorithm for maximum compatibility.

The **Encoding** property also determines the encoding method that is used when the **DecodedText** and **EncodedText** properties are used to decode and encode text strings.

See Also

FileEncoder Class | SocketTools Namespace

FileEncoder.IsInitialized Property

Gets a value which indicates if the current instance of the class has been initialized successfully.

[Visual Basic]

```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]

```
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the class instance has been initialized; otherwise returns **false**.

Remarks

The **IsInitialized** property is used to determine if the current instance of the class has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the class may not be able to initialize itself.

The most common reasons that a class instance may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required networking libraries or not being able to access the system registry.

See Also

FileEncoder Class | SocketTools Namespace

FileEncoder.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public FileEncoder.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

FileEncoder Class | SocketTools Namespace

FileEncoder.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Remarks

A string which describes the last error that has occurred. The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

FileEncoder Class | SocketTools Namespace

FileEncoder.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]

```
Public Property ThrowError As Boolean
```

[C#]

```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

FileEncoder Class | SocketTools Namespace

FileEncoder.Version Property

Gets a value which returns the current version of the FileEncoder class library.

[Visual Basic]

```
Public ReadOnly Property Version As String
```

[C#]

```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the FileEncoder class library. This value can be used by an application for validation and debugging purposes.

See Also

FileEncoder Class | SocketTools Namespace

FileEncoder Methods

The methods of the **FileEncoder** class are listed below. For a complete list of **FileEncoder** class members, see the FileEncoder Members topic.

Public Instance Methods

CompressData	Compresses the contents of the specified buffer.
CompressFile	Overloaded. Compress the contents of the specified file.
DecodeFile	Overloaded. Decodes an encoded file, storing the contents in the specified file.
DecryptData	Overloaded. Decrypts the contents of a byte array.
DecryptFile	Overloaded. Decrypts the contents of a file using AES-256 encryption.
Dispose	Overloaded. Releases all resources used by FileEncoder.
EncodeFile	Overloaded. Encodes a file, storing the contents as printable text in the specified file.
EncryptData	Overloaded. Encrypts the contents of a byte array.
EncryptFile	Overloaded. Encrypts the contents of a file using AES-256 encryption.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
ExpandData	Expands the contents of the compressed buffer.
ExpandFile	Overloaded. Expands the contents of a previously compressed file.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initializes the component with the specified runtime license key.
Reset	Reset the internal state of the class back to its defaults.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the component.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the FileEncoder class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the

	resources allocated for the object.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FileEncoder Class | SocketTools Namespace

FileEncoder.CompressData Method

Compresses the contents of the specified buffer.

[Visual Basic]

```
Public Function CompressData( _
   ByVal inputBuffer As Byte(), _
   ByVal inputLength As Integer, _
   ByVal outputBuffer As Byte(), _
   ByRef outputLength As Integer _
) As Boolean
```

[C#]

```
public bool CompressData(
   byte[] inputBuffer,
   int inputLength,
   byte[] outputBuffer,
   ref int outputLength
);
```

Parameters

inputBuffer
A byte array which contains the data to be compressed.

inputLength
An integer value which specifies the number of bytes to be compressed.

outputBuffer
A byte array which will contain the compressed data.

outputLength
An integer value which specifies the maximum number of bytes that can be copied into the output buffer.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

See Also

FileEncoder Class | SocketTools Namespace | CompressFile | ExpandData | ExpandFile

FileEncoder.CompressFile Method

Compress the contents of the specified file.

Overload List

Compress the contents of the specified file.

public bool CompressFile(string,string);

Compress the contents of the specified file.

public bool CompressFile(string,string,CompressionType);

Compress the contents of the specified file.

public bool CompressFile(string,string,CompressionType,CompressionLevel);

See Also

FileEncoder Class | SocketTools Namespace | CompressData | ExpandData | ExpandFile

FileEncoder.CompressFile Method (String, String)

Compress the contents of the specified file.

[Visual Basic]

```
Overloads Public Function CompressFile( _
   ByVal inputFile As String, _
   ByVal outputFile As String _
) As Boolean
```

[C#]

```
public bool CompressFile(
   string inputFile,
   string outputFile
);
```

Parameters

inputFile

The name of the file that will be compressed. The file must exist, and it must be a regular file that can be opened for reading by the current process. An error will be returned if a character device, such as the console, is specified as the file name.

outputFile

The name of the file that is to contain the compressed file data. If the file exists, it must be a regular file that can be opened for writing by the current process and will be overwritten. If the file does not exist, it will be created. An error will be returned if a character device, such as the console, is specified as the file name.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

It is permitted to specify an empty string as the output file name. In this case, the method will create a temporary file in the directory specified by the TEMP environment variable. After the file has been compressed, the **OutputFile** property will be set the name of the temporary file that has been created. It is the responsibility of the application to delete this temporary file after it is no longer needed.

Note that the compressed file is not stored in an archive format that is recognized by third-party applications such as PKZip or WinZip.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.CompressFile Overload List | CompressData | ExpandData | ExpandFile

FileEncoder.CompressFile Method (String, String, CompressionType)

Compress the contents of the specified file.

[Visual Basic]
```
Overloads Public Function CompressFile( _
   ByVal inputFile As String, _
   ByVal outputFile As String, _
   ByVal compressionType As CompressionType _
) As Boolean
```

[C#]
```
public bool CompressFile(
   string inputFile,
   string outputFile,
   CompressionType compressionType
);
```

Parameters

inputFile
The name of the file that will be compressed. The file must exist, and it must be a regular file that can be opened for reading by the current process. An error will be returned if a character device, such as the console, is specified as the file name.

outputFile
The name of the file that is to contain the compressed file data. If the file exists, it must be a regular file that can be opened for writing by the current process and will be overwritten. If the file does not exist, it will be created. An error will be returned if a character device, such as the console, is specified as the file name.

compressionType
One of the CompressionType enumeration values which determines the algorithm that will be used to compress the data.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

It is permitted to specify an empty string as the output file name. In this case, the method will create a temporary file in the directory specified by the TEMP environment variable. After the file has been compressed, the **OutputFile** property will be set the name of the temporary file that has been created. It is the responsibility of the application to delete this temporary file after it is no longer needed.

Note that the compressed file is not stored in an archive format that is recognized by third-party applications such as PKZip or WinZip.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.CompressFile Overload List | CompressData | ExpandData | ExpandFile

FileEncoder.CompressFile Method (String, String, CompressionType, CompressionLevel)

Compress the contents of the specified file.

[Visual Basic]
```
Overloads Public Function CompressFile( _
   ByVal inputFile As String, _
   ByVal outputFile As String, _
   ByVal compressionType As CompressionType, _
   ByVal compressionLevel As CompressionLevel _
) As Boolean
```

[C#]
```
public bool CompressFile(
   string inputFile,
   string outputFile,
   CompressionType compressionType,
   CompressionLevel compressionLevel
);
```

Parameters

inputFile

The name of the file that will be compressed. The file must exist, and it must be a regular file that can be opened for reading by the current process. An error will be returned if a character device, such as the console, is specified as the file name.

outputFile

The name of the file that is to contain the compressed file data. If the file exists, it must be a regular file that can be opened for writing by the current process and will be overwritten. If the file does not exist, it will be created. An error will be returned if a character device, such as the console, is specified as the file name.

compressionType

One of the CompressionType enumeration values which determines the algorithm that will be used to compress the data.

compressionLevel

One of the CompressionLevel enumeration values which specifies the compression level to use. A value of zero specifies that the default compression level appropriate for the selected algorithm should be used, balancing resource usage and the compression ratio of the data. A value of 1 specifies that the compression should be performed using minimal memory resources, at the expense of the compression ratio. The maximum value of 9 specifies that the algorithm should use more memory to achieve the maximum compression ratio. It is recommended that most applications use the default compression level.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

It is permitted to specify an empty string as the output file name. In this case, the method will create a temporary file in the directory specified by the TEMP environment variable. After the file has been compressed, the **OutputFile** property will be set the name of the temporary file that has been created. It is the responsibility of the application to delete this temporary file after it is no longer needed.

Note that the compressed file is not stored in an archive format that is recognized by third-party applications such as PKZip or WinZip.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.CompressFile Overload List | CompressData | ExpandData | ExpandFile

FileEncoder.DecodeFile Method

Decodes an encoded file, storing the contents in the specified file.

Overload List

Decodes an encoded file, storing the contents in the specified file.

public bool DecodeFile(string,string);

Decodes an encoded file, storing the contents in the specified file.

public bool DecodeFile(string,string,EncodingType);

See Also

FileEncoder Class | SocketTools Namespace | DecryptFile | EncodeFile | EncryptFile

FileEncoder.DecodeFile Method (String, String)

Decodes an encoded file, storing the contents in the specified file.

[Visual Basic]

```
Overloads Public Function DecodeFile( _
   ByVal inputFile As String, _
   ByVal outputFile As String _
) As Boolean
```

[C#]

```
public bool DecodeFile(
   string inputFile,
   string outputFile
);
```

Parameters

inputFile

A string which specifies the name of the file to be decoded. The file must exist, and it must be a regular file that can be opened for reading by the current process. An error will be returned if a character device, such as CON: is specified as the file name.

outputFile

The name of the file that is to contain the decoded file data. If the file exists, it must be a regular file that can be opened for writing by the current process and will be overwritten. If the file does not exist, it will be created. An error will be returned if a character device, such as CON: is specified as the file name.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.DecodeFile Overload List | DecryptFile | EncodeFile | EncryptFile

FileEncoder.DecodeFile Method (String, String, EncodingType)

Decodes an encoded file, storing the contents in the specified file.

[Visual Basic]

```
Overloads Public Function DecodeFile( _
   ByVal inputFile As String, _
   ByVal outputFile As String, _
   ByVal encodingType As EncodingType _
) As Boolean
```

[C#]

```
public bool DecodeFile(
   string inputFile,
   string outputFile,
   EncodingType encodingType
);
```

Parameters

inputFile
A string which specifies the name of the file to be decoded. The file must exist, and it must be a regular file that can be opened for reading by the current process. An error will be returned if a character device, such as CON: is specified as the file name.

outputFile
The name of the file that is to contain the decoded file data. If the file exists, it must be a regular file that can be opened for writing by the current process and will be overwritten. If the file does not exist, it will be created. An error will be returned if a character device, such as CON: is specified as the file name.

encodingType
An EncodingType enumeration value which specifies the type of encoding that was used to encode the data.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.DecodeFile Overload List | DecryptFile | EncodeFile | EncryptFile

FileEncoder.DecryptData Method

Decrypts the contents of a byte array.

Overload List

Decrypts the contents of a byte array.

public bool DecryptData(byte[],int,byte[],ref int);

Decrypts the contents of a **System.IO.MemoryStream** object.

public bool DecryptData(MemoryStream,MemoryStream);

Decrypts the contents of a byte array.

public bool DecryptData(string,byte[],int,byte[],ref int);

Decrypts the contents of a **System.IO.MemoryStream** object.

public bool DecryptData(string,MemoryStream,MemoryStream);

Decrypts the contents of a string.

public bool DecryptData(string,string,ref string);

Decrypts the contents of a string.

public bool DecryptData(string,ref string);

See Also

FileEncoder Class | SocketTools Namespace | DecryptFile | EncryptData | EncryptFile

FileEncoder.DecryptData Method (String, Byte[], Int32, Byte[], Int32)

Decrypts the contents of a byte array.

[Visual Basic]

```
Overloads Public Function DecryptData( _
   ByVal password As String, _
   ByVal inputBuffer As Byte(), _
   ByVal inputSize As Integer, _
   ByVal outputBuffer As Byte(), _
   ByRef outputSize As Integer _
) As Boolean
```

[C#]

```
public bool DecryptData(
   string password,
   byte[] inputBuffer,
   int inputSize,
   byte[] outputBuffer,
   ref int outputSize
);
```

Parameters

password
A string of characters that will be used to generate the decryption key. This parameter may be a zero-length string, in which case a default internal hash value is used to decrypt the data. Password strings that exceed 215 characters will be truncated.

inputBuffer
A byte array which contains the data to be decrypted.

inputSize
An integer value which specifies the number of bytes in the byte array which contains the encrypted data.

outputBuffer
A byte array which will contain the decrypted data when the method returns.

outputSize
An integer value which specifies the maximum number of bytes that can be stored in the output byte array. When the method returns, this parameter will contain the number of decrypted bytes in the buffer.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

This method will decrypt the contents of a byte array using a 256-bit AES (Advanced Encryption Standard) algorithm and returns a copy of the decrypted data to the caller. The password (or passphrase) provided by the caller is used to generate a SHA-256 hash value which is used as part of the decryption process. The password value must be identical to the value used to encrypt the data using the **EncryptData** method.

Due to how the SHA-256 hash is generated, this method cannot be used to decrypt data that was encrypted using another third-party library. It can only be used to decrypt data that was previously encrypted using **EncryptData**.

If you wish to decrypt the contents of a file, use the **DecryptFile** method.

This method uses the Microsoft CryptoAPI and the RSA AES cryptographic provider. This provider may not be available in some languages, countries or regions. The availability of this provider may also be constrained by cryptography export restrictions imposed by the United States or other countries. If the required cryptographic provider is not available, the method will fail.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.DecryptData Overload List | DecryptFile | EncryptData | EncryptFile

FileEncoder.DecryptData Method (Byte[], Int32, Byte[], Int32)

Decrypts the contents of a byte array.

[Visual Basic]

```
Overloads Public Function DecryptData( _
   ByVal inputBuffer As Byte(), _
   ByVal inputSize As Integer, _
   ByVal outputBuffer As Byte(), _
   ByRef outputSize As Integer _
) As Boolean
```

[C#]

```
public bool DecryptData(
   byte[] inputBuffer,
   int inputSize,
   byte[] outputBuffer,
   ref int outputSize
);
```

Parameters

inputBuffer
A byte array which contains the data to be decrypted.

inputSize
An integer value which specifies the number of bytes in the byte array which contains the encrypted data.

outputBuffer
A byte array which will contain the decrypted data when the method returns.

outputSize
An integer value which specifies the maximum number of bytes that can be stored in the output byte array. When the method returns, this parameter will contain the number of decrypted bytes in the buffer.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

This method will decrypt the contents of a byte array using a 256-bit AES (Advanced Encryption Standard) algorithm and returns a copy of the decrypted data to the caller. The value of the **Password** property is used to generate a SHA-256 hash value which is used as part of the decryption process. The password value must be identical to the value used to encrypt the data using the **EncryptData** method.

Due to how the SHA-256 hash is generated, this method cannot be used to decrypt data that was encrypted using another third-party library. It can only be used to decrypt data that was previously encrypted using **EncryptData**.

If you wish to decrypt the contents of a file, use the **DecryptFile** method.

This method uses the Microsoft CryptoAPI and the RSA AES cryptographic provider. This provider may not be available in some languages, countries or regions. The availability of this provider may also be constrained by cryptography export restrictions imposed by the United States or other countries. If the required cryptographic provider is not available, the method will fail.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.DecryptData Overload List | DecryptFile | EncryptData | EncryptFile

FileEncoder.DecryptData Method (String, MemoryStream, MemoryStream)

Decrypts the contents of a **System.IO.MemoryStream** object.

```
[Visual Basic]
Overloads Public Function DecryptData( _
   ByVal password As String, _
   ByVal inputStream As MemoryStream, _
   ByVal outputStream As MemoryStream _
) As Boolean
```

```
[C#]
public bool DecryptData(
   string password,
   MemoryStream inputStream,
   MemoryStream outputStream
);
```

Parameters

password
A string of characters that will be used to generate the decryption key. This parameter may be a zero-length string, in which case a default internal hash value is used to decrypt the data. Password strings that exceed 215 characters will be truncated.

inputStream
A **MemoryStream** object which contains the data to be decrypted. The stream must be readable.

outputStream
A **MemoryStream** object which will contain the decrypted data when the method returns. The caller must create a stream that is writable, and the contents of the stream will be replaced with the decrypted data.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

This method will decrypt the contents of a byte array using a 256-bit AES (Advanced Encryption Standard) algorithm and returns a copy of the decrypted data to the caller. The password (or passphrase) provided by the caller is used to generate a SHA-256 hash value which is used as part of the decryption process. The password value must be identical to the value used to encrypt the data using the **EncryptData** method.

Due to how the SHA-256 hash is generated, this method cannot be used to decrypt data that was encrypted using another third-party library. It can only be used to decrypt data that was previously encrypted using **EncryptData**.

If you wish to decrypt the contents of a file, use the **DecryptFile** method.

This method uses the Microsoft CryptoAPI and the RSA AES cryptographic provider. This provider may not be available in some languages, countries or regions. The availability of this provider may also be constrained by cryptography export restrictions imposed by the United States or other countries. If the required cryptographic provider is not available, the method will fail.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.DecryptData Overload List | DecryptFile | EncryptData | EncryptFile

FileEncoder.DecryptData Method (MemoryStream, MemoryStream)

Decrypts the contents of a **System.IO.MemoryStream** object.

[Visual Basic]

```
Overloads Public Function DecryptData( _
   ByVal inputStream As MemoryStream, _
   ByVal outputStream As MemoryStream _
) As Boolean
```

[C#]

```
public bool DecryptData(
   MemoryStream inputStream,
   MemoryStream outputStream
);
```

Parameters

inputStream

A **MemoryStream** object which contains the data to be decrypted. The stream must be readable.

outputStream

A **MemoryStream** object which will contain the decrypted data when the method returns. The caller must create a stream that is writable, and the contents of the stream will be replaced with the decrypted data.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

This method will decrypt the contents of a byte array using a 256-bit AES (Advanced Encryption Standard) algorithm and returns a copy of the decrypted data to the caller. The value of the **Password** property is used to generate a SHA-256 hash value which is used as part of the decryption process. The password value must be identical to the value used to encrypt the data using the **EncryptData** method.

Due to how the SHA-256 hash is generated, this method cannot be used to decrypt data that was encrypted using another third-party library. It can only be used to decrypt data that was previously encrypted using **EncryptData**.

If you wish to decrypt the contents of a file, use the **DecryptFile** method.

This method uses the Microsoft CryptoAPI and the RSA AES cryptographic provider. This provider may not be available in some languages, countries or regions. The availability of this provider may also be constrained by cryptography export restrictions imposed by the United States or other countries. If the required cryptographic provider is not available, the method will fail.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.DecryptData Overload List | DecryptFile | EncryptData | EncryptFile

FileEncoder.DecryptData Method (String, String, String)

Decrypts the contents of a string.

[Visual Basic]

```
Overloads Public Function DecryptData( _
   ByVal password As String, _
   ByVal inputText As String, _
   ByRef outputText As String _
) As Boolean
```

[C#]

```
public bool DecryptData(
   string password,
   string inputText,
   ref string outputText
);
```

Parameters

password

A string of characters that will be used to generate the decryption key. This parameter may be a zero-length string, in which case a default internal hash value is used to decrypt the data. Password strings that exceed 215 characters will be truncated.

inputText

A string which contains the encoded data to be decrypted.

outputText

A string which will contain the decrypted text when the method returns.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

This method will decrypt the contents of a string using a 256-bit AES (Advanced Encryption Standard) algorithm and returns a copy of the decrypted text to the caller. The password (or passphrase) provided by the caller is used to generate a SHA-256 hash value which is used as part of the decryption process. The password value must be identical to the value used to encrypt the data using the **EncryptData** method.

Due to how the SHA-256 hash is generated, this method cannot be used to decrypt text that was encrypted using another third-party library. It can only be used to decrypt text that was previously encrypted using **EncryptData**.

If you wish to decrypt the contents of a file, use the **DecryptFile** method.

This method uses the Microsoft CryptoAPI and the RSA AES cryptographic provider. This provider may not be available in some languages, countries or regions. The availability of this provider may also be constrained by cryptography export restrictions imposed by the United States or other countries. If the required cryptographic provider is not available, the method will fail.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.DecryptData Overload List | DecryptFile |

EncryptFile | EncryptData

FileEncoder.DecryptData Method (String, String)

Decrypts the contents of a string.

[Visual Basic]

```
Overloads Public Function DecryptData( _
   ByVal inputText As String, _
   ByRef outputText As String _
) As Boolean
```

[C#]

```
public bool DecryptData(
   string inputText,
   ref string outputText
);
```

Parameters

inputText
A string which contains the encoded data to be decrypted.

outputText
A string which will contain the decrypted text when the method returns.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

This method will decrypt the contents of a string using a 256-bit AES (Advanced Encryption Standard) algorithm and returns a copy of the decrypted text to the caller. The value of the **Password** property is used to generate a SHA-256 hash value which is used as part of the decryption process. The password value must be identical to the value used to encrypt the data using the **EncryptData** method.

Due to how the SHA-256 hash is generated, this method cannot be used to decrypt text that was encrypted using another third-party library. It can only be used to decrypt text that was previously encrypted using **EncryptData**.

If you wish to decrypt the contents of a file, use the **DecryptFile** method.

This method uses the Microsoft CryptoAPI and the RSA AES cryptographic provider. This provider may not be available in some languages, countries or regions. The availability of this provider may also be constrained by cryptography export restrictions imposed by the United States or other countries. If the required cryptographic provider is not available, the method will fail.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.DecryptData Overload List | EncryptData | EncryptFile

FileEncoder.DecryptFile Method

Decrypts the contents of a file using AES-256 encryption.

Overload List

Decrypts the contents of a file using AES-256 encryption.

public bool DecryptFile(string,string);

Decrypts the contents of a file using AES-256 encryption.

public bool DecryptFile(string,string,string);

See Also

FileEncoder Class | SocketTools Namespace | DecryptData | EncryptData | EncryptFile

FileEncoder.DecryptFile Method (String, String, String)

Decrypts the contents of a file using AES-256 encryption.

[Visual Basic]

```
Overloads Public Function DecryptFile( _
   ByVal password As String, _
   ByVal inputFile As String, _
   ByVal outputFile As String _
) As Boolean
```

[C#]

```
public bool DecryptFile(
   string password,
   string inputFile,
   string outputFile
);
```

Parameters

password

A string of characters that will be used to generate the decryption key. Passwords are case sensitive and must match exactly when decrypting data that was previously encrypted using this method. This parameter may be a zero-length string, in which case a default internal hash value is used. Password strings that exceed 215 characters will be truncated.

inputFile

A string which specifies the name of the file to be decrypted. The file must exist, and it must be a regular file that can be opened for reading by the current process. An error will be returned if a character device, such as CON: is specified as the file name.

outputFile

The name of the file that is to contain the decrypted file data. If the file exists, it must be a regular file that can be opened for writing by the current process and will be overwritten. If the file does not exist, it will be created. An error will be returned if a character device, such as CON: is specified as the file name.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

This method will decrypt the contents of a file using a 256-bit AES (Advanced Encryption Standard) algorithm and stores the decrypted data in the specified output file. The password (or passphrase) provided by the caller is used to generate a SHA-256 hash value which is used as part of the decryption process. The password must be identical to the value used to encrypt the data using the **EncryptFile** method.

Due to how the SHA-256 hash is generated, this method cannot be used to decrypt files that were encrypted using another third-party library. It can only be used to decrypt data that was previously encrypted using the **EncryptFile** method.

A temporary file is created during the decryption process and the output file is created or overwritten only if the input file could be successfully encrypted. If the decryption fails, no output file will be created.

If you wish to decrypt the contents of a memory buffer or string, use the **DecryptData** method.

This method uses the Microsoft CryptoAPI and the RSA AES cryptographic provider. This provider may not be available in some languages, countries or regions. The availability of this provider may also be constrained by cryptography export restrictions imposed by the United States or other countries. If the required cryptographic provider is not available, the method will fail.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.DecryptFile Overload List | DecryptData | EncryptData | EncryptFile

FileEncoder.DecryptFile Method (String, String)

Decrypts the contents of a file using AES-256 encryption.

[Visual Basic]

```
Overloads Public Function DecryptFile( _
   ByVal inputFile As String, _
   ByVal outputFile As String _
) As Boolean
```

[C#]

```
public bool DecryptFile(
   string inputFile,
   string outputFile
);
```

Parameters

inputFile

A string which specifies the name of the file to be decrypted. The file must exist, and it must be a regular file that can be opened for reading by the current process. An error will be returned if a character device, such as CON: is specified as the file name.

outputFile

The name of the file that is to contain the decrypted file data. If the file exists, it must be a regular file that can be opened for writing by the current process and will be overwritten. If the file does not exist, it will be created. An error will be returned if a character device, such as CON: is specified as the file name.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

This method will decrypt the contents of a file using a 256-bit AES (Advanced Encryption Standard) algorithm and stores the decrypted data in the specified output file. The value of the **Password** property is used to generate a SHA-256 hash value which is used as part of the decryption process. The password must be identical to the value used to encrypt the data using the **EncryptFile** method.

Due to how the SHA-256 hash is generated, this method cannot be used to decrypt files that were encrypted using another third-party library. It can only be used to decrypt data that was previously encrypted using the **EncryptFile** method.

A temporary file is created during the decryption process and the output file is created or overwritten only if the input file could be successfully encrypted. If the decryption fails, no output file will be created.

If you wish to decrypt the contents of a memory buffer or string, use the **DecryptData** method.

This method uses the Microsoft CryptoAPI and the RSA AES cryptographic provider. This provider may not be available in some languages, countries or regions. The availability of this provider may also be constrained by cryptography export restrictions imposed by the United States or other countries. If the required cryptographic provider is not available, the method will fail.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.DecryptFile Overload List | DecryptData |

EncryptData | EncryptFile

FileEncoder.EncodeFile Method

Encodes a file, storing the contents as printable text in the specified file.

Overload List

Encodes a file, storing the contents as printable text in the specified file.

public bool EncodeFile(string,string);

Encodes a file, storing the contents as printable text in the specified file.

public bool EncodeFile(string,string,EncodingType);

See Also

FileEncoder Class | SocketTools Namespace | DecodeFile

FileEncoder.EncodeFile Method (String, String)

Encodes a file, storing the contents as printable text in the specified file.

[Visual Basic]

```
Overloads Public Function EncodeFile( _
   ByVal inputFile As String, _
   ByVal outputFile As String _
) As Boolean
```

[C#]

```
public bool EncodeFile(
   string inputFile,
   string outputFile
);
```

Parameters

inputFile

A string which specifies the name of the file to be encoded. The file must exist, and it must be a regular file that can be opened for reading by the current process. An error will be returned if a character device, such as CON: is specified as the file name.

outputFile

The name of the file that is to contain the encoded file data. If the file exists, it must be a regular file that can be opened for writing by the current process and will be overwritten. If the file does not exist, it will be created. An error will be returned if a character device, such as CON: is specified as the file name.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.EncodeFile Overload List | DecodeFile

FileEncoder.EncodeFile Method (String, String, EncodingType)

Encodes a file, storing the contents as printable text in the specified file.

[Visual Basic]

```
Overloads Public Function EncodeFile( _
   ByVal inputFile As String, _
   ByVal outputFile As String, _
   ByVal encodingType As EncodingType _
) As Boolean
```

[C#]

```
public bool EncodeFile(
   string inputFile,
   string outputFile,
   EncodingType encodingType
);
```

Parameters

inputFile
: A string which specifies the name of the file to be encoded. The file must exist, and it must be a regular file that can be opened for reading by the current process. An error will be returned if a character device, such as CON: is specified as the file name.

outputFile
: The name of the file that is to contain the encoded file data. If the file exists, it must be a regular file that can be opened for writing by the current process and will be overwritten. If the file does not exist, it will be created. An error will be returned if a character device, such as CON: is specified as the file name.

encodingType
: An EncodingType enumeration value which specifies the type of encoding which should be used.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.EncodeFile Overload List | DecodeFile

FileEncoder.EncryptData Method

Encrypts the contents of a byte array.

Overload List

Encrypts the contents of a byte array.

public bool EncryptData(byte[],int,byte[],ref int);

Encrypts the contents of a **System.IO.MemoryStream** object.

public bool EncryptData(MemoryStream,MemoryStream);

Encrypts the contents of a byte array.

public bool EncryptData(string,byte[],int,byte[],ref int);

Encrypts the contents of a **System.IO.MemoryStream** object.

public bool EncryptData(string,MemoryStream,MemoryStream);

Encrypts the contents of a string.

public bool EncryptData(string,string,ref string);

Encrypts the contents of a string.

public bool EncryptData(string,ref string);

See Also

FileEncoder Class | SocketTools Namespace | DecryptData | DecryptFile | EncryptFile

FileEncoder.EncryptData Method (String, Byte[], Int32, Byte[], Int32)

Encrypts the contents of a byte array.

[Visual Basic]

```
Overloads Public Function EncryptData( _
   ByVal password As String, _
   ByVal inputBuffer As Byte(), _
   ByVal inputSize As Integer, _
   ByVal outputBuffer As Byte(), _
   ByRef outputSize As Integer _
) As Boolean
```

[C#]

```
public bool EncryptData(
   string password,
   byte[] inputBuffer,
   int inputSize,
   byte[] outputBuffer,
   ref int outputSize
);
```

Parameters

password

A string of characters that will be used to generate the encryption key. This parameter may be a zero-length string, in which case a default internal hash value is used to encrypt the data. Password strings that exceed 215 characters will be truncated.

inputBuffer

A byte array which contains the data to be encrypted.

inputSize

An integer value which specifies the number of bytes in the byte array which contains the unencrypted data.

outputBuffer

A byte array which will contain the encrypted data when the method returns.

outputSize

An integer value which specifies the maximum number of bytes that can be stored in the output byte array. When the method returns, this parameter will contain the number of encrypted bytes of data in the buffer.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

This method will encrypt the contents of a byte array using a 256-bit AES (Advanced Encryption Standard) algorithm and returns a copy of the encrypted data to the caller. The password (or passphrase) provided by the caller is used to generate a SHA-256 hash value which is used as part of the encryption process.

It is recommended that most applications specify a password value. If the password is a zero-length string, a default internal hash value is used. This means that any other application which uses SocketTools will be able to decrypt the data.

The amount of encrypted data returned by this method will always be somewhat larger than original unencrypted data. If your application dynamically allocates a block of memory to store the encrypted data, provide a maximum buffer size that is at least several hundred bytes larger than the unencrypted data. If the output buffer provided is not large enough, the method will fail.

The encrypted data returned by this method can contain embedded null bytes. If you wish to encrypt strings and store the encrypted values as printable text, use the version of this method which accepts string parameters. It will perform the same 256-bit AES encryption, but return the encrypted data as a base64 encoded string rather than binary data.

Due to how the SHA-256 hash is generated, the encrypted data cannot be decrypted using another third-party library with the same password value. It can only be decrypted using the **DecryptData** method.

If you wish to encrypt the contents of a file, use the **EncryptFile** method.

This method uses the Microsoft CryptoAPI and the RSA AES cryptographic provider. This provider may not be available in some languages, countries or regions. The availability of this provider may also be constrained by cryptography export restrictions imposed by the United States or other countries. If the required cryptographic provider is not available, the method will fail.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.EncryptData Overload List | DecryptData | DecryptFile | EncryptFile

FileEncoder.EncryptData Method (Byte[], Int32, Byte[], Int32)

Encrypts the contents of a byte array.

[Visual Basic]

```
Overloads Public Function EncryptData( _
   ByVal inputBuffer As Byte(), _
   ByVal inputSize As Integer, _
   ByVal outputBuffer As Byte(), _
   ByRef outputSize As Integer _
) As Boolean
```

[C#]

```
public bool EncryptData(
   byte[] inputBuffer,
   int inputSize,
   byte[] outputBuffer,
   ref int outputSize
);
```

Parameters

inputBuffer
A byte array which contains the data to be encrypted.

inputSize
An integer value which specifies the number of bytes in the byte array which contains the unencrypted data.

outputBuffer
A byte array which will contain the encrypted data when the method returns.

outputSize
An integer value which specifies the maximum number of bytes that can be stored in the output byte array. When the method returns, this parameter will contain the number of encrypted bytes of data in the buffer.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

This method will encrypt the contents of a byte array using a 256-bit AES (Advanced Encryption Standard) algorithm and returns a copy of the encrypted data to the caller. The value of the **Password** property is used to generate a SHA-256 hash value which is used as part of the encryption process.

It is recommended that most applications specify a password value. If the password is a zero-length string, a default internal hash value is used. This means that any other application which uses SocketTools will be able to decrypt the data.

The amount of encrypted data returned by this method will always be somewhat larger than original unencrypted data. If your application dynamically allocates a block of memory to store the encrypted data, provide a maximum buffer size that is at least several hundred bytes larger than the unencrypted data. If the output buffer provided is not large enough, the method will fail.

The encrypted data returned by this method can contain embedded null bytes. If you wish to encrypt strings and store the encrypted values as printable text, use the version of this method which accepts string parameters. It will perform the same 256-bit AES encryption, but return the encrypted data as a base64 encoded string rather than binary data.

Due to how the SHA-256 hash is generated, the encrypted data cannot be decrypted using another third-party library with the same password value. It can only be decrypted using the **DecryptData** method.

If you wish to encrypt the contents of a file, use the **EncryptFile** method.

This method uses the Microsoft CryptoAPI and the RSA AES cryptographic provider. This provider may not be available in some languages, countries or regions. The availability of this provider may also be constrained by cryptography export restrictions imposed by the United States or other countries. If the required cryptographic provider is not available, the method will fail.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.EncryptData Overload List | DecryptData | DecryptFile | EncryptFile

FileEncoder.EncryptData Method (String, MemoryStream, MemoryStream)

Encrypts the contents of a **System.IO.MemoryStream** object.

[Visual Basic]

```
Overloads Public Function EncryptData( _
   ByVal password As String, _
   ByVal inputStream As MemoryStream, _
   ByVal outputStream As MemoryStream _
) As Boolean
```

[C#]

```
public bool EncryptData(
   string password,
   MemoryStream inputStream,
   MemoryStream outputStream
);
```

Parameters

password
A string of characters that will be used to generate the encryption key. This parameter may be a zero-length string, in which case a default internal hash value is used to encrypt the data. Password strings that exceed 215 characters will be truncated.

inputStream
A **MemoryStream** object which contains the data to be encrypted. The stream must be readable.

outputStream
A **MemoryStream** object which will contain the encrypted data when the method returns. The caller must create a stream that is writable, and the contents of the stream will be replaced with the encrypted data.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

This method will encrypt the contents of a **MemoryStream** object using a 256-bit AES (Advanced Encryption Standard) algorithm and returns a copy of the encrypted data to the caller. The password (or passphrase) provided by the caller is used to generate a SHA-256 hash value which is used as part of the encryption process.

It is recommended that most applications specify a password value. If the password is a zero-length string, a default internal hash value is used. This means that any other application which uses SocketTools will be able to decrypt the data.

The amount of encrypted data returned by this method will always be somewhat larger than original unencrypted data. If your application dynamically allocates a block of memory to store the encrypted data, provide a maximum buffer size that is at least several hundred bytes larger than the unencrypted data. If the output buffer provided is not large enough, the method will fail.

The encrypted data returned by this method can contain embedded null bytes. If you wish to encrypt strings and store the encrypted values as printable text, use the version of this method which accepts string parameters. It will perform the same 256-bit AES encryption, but return the encrypted data as a base64 encoded string rather than binary data.

Due to how the SHA-256 hash is generated, the encrypted data cannot be decrypted using another third-party library with the same password value. It can only be decrypted using the **DecryptData** method.

If you wish to encrypt the contents of a file, use the **EncryptFile** method.

This method uses the Microsoft CryptoAPI and the RSA AES cryptographic provider. This provider may not be available in some languages, countries or regions. The availability of this provider may also be constrained by cryptography export restrictions imposed by the United States or other countries. If the required cryptographic provider is not available, the method will fail.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.EncryptData Overload List | DecryptData | DecryptFile | EncryptFile

FileEncoder.EncryptData Method (MemoryStream, MemoryStream)

Encrypts the contents of a **System.IO.MemoryStream** object.

[Visual Basic]

```
Overloads Public Function EncryptData( _
   ByVal inputStream As MemoryStream, _
   ByVal outputStream As MemoryStream _
) As Boolean
```

[C#]

```
public bool EncryptData(
   MemoryStream inputStream,
   MemoryStream outputStream
);
```

Parameters

inputStream

A **MemoryStream** object which contains the data to be encrypted. The stream must be readable.

outputStream

A **MemoryStream** object which will contain the encrypted data when the method returns. The caller must create a stream that is writable, and the contents of the stream will be replaced with the encrypted data.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

This method will encrypt the contents of a **MemoryStream** object using a 256-bit AES (Advanced Encryption Standard) algorithm and returns a copy of the encrypted data to the caller. The value of the **Password** property is used to generate a SHA-256 hash value which is used as part of the encryption process.

It is recommended that most applications specify a password value. If the password is a zero-length string, a default internal hash value is used. This means that any other application which uses SocketTools will be able to decrypt the data.

The amount of encrypted data returned by this method will always be somewhat larger than original unencrypted data. If your application dynamically allocates a block of memory to store the encrypted data, provide a maximum buffer size that is at least several hundred bytes larger than the unencrypted data. If the output buffer provided is not large enough, the method will fail.

The encrypted data returned by this method can contain embedded null bytes. If you wish to encrypt strings and store the encrypted values as printable text, use the version of this method which accepts string parameters. It will perform the same 256-bit AES encryption, but return the encrypted data as a base64 encoded string rather than binary data.

Due to how the SHA-256 hash is generated, the encrypted data cannot be decrypted using another third-party library with the same password value. It can only be decrypted using the **DecryptData** method.

If you wish to encrypt the contents of a file, use the **EncryptFile** method.

This method uses the Microsoft CryptoAPI and the RSA AES cryptographic provider. This provider may not be available in some languages, countries or regions. The availability of this provider may also be constrained by cryptography export restrictions imposed by the United States or other countries. If the required cryptographic provider is not available, the method will fail.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.EncryptData Overload List | DecryptData | DecryptFile | EncryptFile

FileEncoder.EncryptData Method (String, String, String)

Encrypts the contents of a string.

[Visual Basic]

```
Overloads Public Function EncryptData( _
   ByVal password As String, _
   ByVal inputText As String, _
   ByRef outputText As String _
) As Boolean
```

[C#]

```
public bool EncryptData(
   string password,
   string inputText,
   ref string outputText
);
```

Parameters

password

A string of characters that will be used to generate the encryption key. This parameter may be a zero-length string, in which case a default internal hash value is used to encrypt the data. Password strings that exceed 215 characters will be truncated.

inputText

A string which contains the text to be encrypted.

outputText

A string which contains the base64 encoded encrypted text.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

This method will encrypt a string using a 256-bit AES (Advanced Encryption Standard) algorithm and returns a copy of the encrypted data as a base64 encoded string to the caller. The password (or passphrase) provided by the caller is used to generate a SHA-256 hash value which is used as part of the encryption process.

It is recommended that most applications specify a password value. If the password is a zero-length string, a default internal hash value is used. This means that any other application which uses SocketTools will be able to decrypt the text.

Due to how the SHA-256 hash is generated, the encrypted text cannot be decrypted using another third-party library with the same password value. It can only be decrypted using the **DecryptData** method.

The plaintext string provided to this method cannot contain embedded nulls and and should not be used to encrypt binary data. If you wish to encrypt binary data, use the version of this method which accepts byte arrays. It will perform the same 256-bit AES encryption and return the encrypted data into a byte array provided by the caller.

If you wish to encrypt the contents of a file, use the **EncryptFile** method.

This method uses the Microsoft CryptoAPI and the RSA AES cryptographic provider. This provider may not be available in some languages, countries or regions. The availability of this provider may also be constrained by cryptography export restrictions imposed by the United States or other countries. If the required cryptographic provider is not available, the method will fail.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.EncryptData Overload List | DecryptData | DecryptFile | EncryptFile

FileEncoder.EncryptData Method (String, String)

Encrypts the contents of a string.

```
[Visual Basic]
Overloads Public Function EncryptData( _
   ByVal inputText As String, _
   ByRef outputText As String _
) As Boolean
```

```
[C#]
public bool EncryptData(
   string inputText,
   ref string outputText
);
```

Parameters

inputText
A string which contains the text to be encrypted.

outputText
A string which contains the base64 encoded encrypted text.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

This method will encrypt a string using a 256-bit AES (Advanced Encryption Standard) algorithm and returns a copy of the encrypted data as a base64 encoded string to the caller. The value of the **Password** property is used to generate a SHA-256 hash value which is used as part of the encryption process.

It is recommended that most applications specify a password value. If the password is a zero-length string, a default internal hash value is used. This means that any other application which uses SocketTools will be able to decrypt the text.

Due to how the SHA-256 hash is generated, the encrypted text cannot be decrypted using another third-party library with the same password value. It can only be decrypted using the **DecryptData** method.

The plaintext string provided to this method cannot contain embedded nulls and and should not be used to encrypt binary data. If you wish to encrypt binary data, use the version of this method which accepts byte arrays. It will perform the same 256-bit AES encryption and return the encrypted data into a byte array provided by the caller.

If you wish to encrypt the contents of a file, use the **EncryptFile** method.

This method uses the Microsoft CryptoAPI and the RSA AES cryptographic provider. This provider may not be available in some languages, countries or regions. The availability of this provider may also be constrained by cryptography export restrictions imposed by the United States or other countries. If the required cryptographic provider is not available, the method will fail.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.EncryptData Overload List | DecryptData | DecryptFile | EncryptFile

FileEncoder.EncryptFile Method

Encrypts the contents of a file using AES-256 encryption.

Overload List

Encrypts the contents of a file using AES-256 encryption.

public bool EncryptFile(string,string);

Encrypts the contents of a file using AES-256 encryption.

public bool EncryptFile(string,string,string);

See Also

FileEncoder Class | SocketTools Namespace | DecryptData | DecryptFile | EncryptData

FileEncoder.EncryptFile Method (String, String, String)

Encrypts the contents of a file using AES-256 encryption.

[Visual Basic]

```
Overloads Public Function EncryptFile( _
   ByVal password As String, _
   ByVal inputFile As String, _
   ByVal outputFile As String _
) As Boolean
```

[C#]

```
public bool EncryptFile(
   string password,
   string inputFile,
   string outputFile
);
```

Parameters

password
A string of characters that will be used to generate the encryption key. Passwords are case sensitive and must match exactly when decrypting data that was previously encrypted using this method. This parameter may be a zero-length string, in which case a default internal hash value is used. Password strings that exceed 215 characters will be truncated.

inputFile
A string which specifies the name of the file to be encrypted. The file must exist, and it must be a regular file that can be opened for reading by the current process. An error will be returned if a character device, such as CON: is specified as the file name.

outputFile
The name of the file that is to contain the encrypted file data. If the file exists, it must be a regular file that can be opened for writing by the current process and will be overwritten. If the file does not exist, it will be created. An error will be returned if a character device, such as CON: is specified as the file name.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

This method will encrypt the contents of a file using a 256-bit AES (Advanced Encryption Standard) algorithm and stores the encrypted data in the specified output file.

It is recommended that most applications specify as password value. If the ***password*** parameter is an empty string, a default internal hash value will be used to create the encryption key. This means that any other SocketTools application which uses this method will be able to decrypt the data.

Due to how the SHA-256 hash is generated, the encrypted data cannot be decrypted using another third-party library with the same password value. It can only be decrypted using the **DecryptFile** method.

A temporary file is created during the encryption process and the output file is created or overwritten only if the input file could be successfully encrypted. If the encryption fails, no output file will be created.

The input file contents will always be processed as a binary data stream. If you use this method to encrypt a text file, the output file will contain binary characters, not printable text. If you wish to transfer or store the encrypted data as text, it should be encoded using the **EncodeFile** method after calling this method.

If your application is also using the **CompressFile** method to compress the data, it is recommended that you call **CompressFile** before calling **EncryptFile**. You will typically achieve a better compression rate on unencrypted data than than attempting to compress data which has been encrypted with this method.

If you wish to encrypt the contents of a memory buffer or string, use the **EncryptData** methods.

This method uses the Microsoft CryptoAPI and the RSA AES cryptographic provider. This provider may not be available in some languages, countries or regions. The availability of this provider may also be constrained by cryptography export restrictions imposed by the United States or other countries. If the required cryptographic provider is not available, the method will fail.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.EncryptFile Overload List | DecryptData | DecryptFile | EncryptData

FileEncoder.EncryptFile Method (String, String)

Encrypts the contents of a file using AES-256 encryption.

[Visual Basic]

```
Overloads Public Function EncryptFile( _
   ByVal inputFile As String, _
   ByVal outputFile As String _
) As Boolean
```

[C#]

```
public bool EncryptFile(
   string inputFile,
   string outputFile
);
```

Parameters

inputFile

A string which specifies the name of the file to be encrypted. The file must exist, and it must be a regular file that can be opened for reading by the current process. An error will be returned if a character device, such as CON: is specified as the file name.

outputFile

The name of the file that is to contain the encrypted file data. If the file exists, it must be a regular file that can be opened for writing by the current process and will be overwritten. If the file does not exist, it will be created. An error will be returned if a character device, such as CON: is specified as the file name.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

Remarks

This method will encrypt the contents of a file using a 256-bit AES (Advanced Encryption Standard) algorithm and stores the encrypted data in the specified output file. The value of the **Password** property is used to generate the encryption key.

It is recommended that most applications specify as password value. If the value of the **Password** property is an empty string, a default internal hash value will be used to create the encryption key. This means that any other SocketTools application which uses this method will be able to decrypt the data.

Due to how the SHA-256 hash is generated, the encrypted data cannot be decrypted using another third-party library with the same password value. It can only be decrypted using the **DecryptFile** method.

A temporary file is created during the encryption process and the output file is created or overwritten only if the input file could be successfully encrypted. If the encryption fails, no output file will be created.

The input file contents will always be processed as a binary data stream. If you use this method to encrypt a text file, the output file will contain binary characters, not printable text. If you wish to transfer or store the encrypted data as text, it should be encoded using the **EncodeFile** method after calling this method.

If your application is also using the **CompressFile** method to compress the data, it is recommended that you call **CompressFile** before calling **EncryptFile**. You will typically achieve a better compression rate on unencrypted data than than attempting to compress data which has been encrypted with this method.

If you wish to encrypt the contents of a memory buffer or string, use the **EncryptData** methods.

This method uses the Microsoft CryptoAPI and the RSA AES cryptographic provider. This provider may not be available in some languages, countries or regions. The availability of this provider may also be constrained by cryptography export restrictions imposed by the United States or other countries. If the required cryptographic provider is not available, the method will fail.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.EncryptFile Overload List | DecryptData | DecryptFile | EncryptData

FileEncoder.ExpandData Method

Expands the contents of the compressed buffer.

[Visual Basic]

```
Public Function ExpandData( _
   ByVal inputBuffer As Byte(), _
   ByVal inputLength As Integer, _
   ByVal outputBuffer As Byte(), _
   ByRef outputLength As Integer _
) As Boolean
```

[C#]

```
public bool ExpandData(
   byte[] inputBuffer,
   int inputLength,
   byte[] outputBuffer,
   ref int outputLength
);
```

Parameters

inputBuffer
A byte array which contains the compressed data to be expanded.

inputLength
An integer value which specifies the number of bytes in the input buffer.

outputBuffer
A byte array which will contain the expanded data.

outputLength
An integer value which specifies the maximum number of bytes that can be copied into the output buffer.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

See Also

FileEncoder Class | SocketTools Namespace | CompressData | CompressFile | ExpandFile

FileEncoder.ExpandFile Method

Expands the contents of a previously compressed file.

Overload List

Expands the contents of a previously compressed file.

public bool ExpandFile(string,string);

Expands the contents of a previously compressed file.

public bool ExpandFile(string,string,CompressionType);

See Also

FileEncoder Class | SocketTools Namespace | CompressFile | EncodeFile | EncryptFile

FileEncoder.ExpandFile Method (String, String)

Expands the contents of a previously compressed file.

[Visual Basic]

```
Overloads Public Function ExpandFile( _
   ByVal inputFile As String, _
   ByVal outputFile As String _
) As Boolean
```

[C#]

```
public bool ExpandFile(
   string inputFile,
   string outputFile
);
```

Parameters

inputFile

A string which specifies the name of the file to be compressed. The file must exist, and it must be a regular file that can be opened for reading by the current process. An error will be returned if a character device, such as CON: is specified as the file name.

outputFile

The name of the file that is to contain the expanded file data. If the file exists, it must be a regular file that can be opened for writing by the current process and will be overwritten. If the file does not exist, it will be created. An error will be returned if a character device, such as CON: is specified as the file name.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.ExpandFile Overload List | CompressFile | EncodeFile | EncryptFile

FileEncoder.ExpandFile Method (String, String, CompressionType)

Expands the contents of a previously compressed file.

[Visual Basic]

```
Overloads Public Function ExpandFile( _
   ByVal inputFile As String, _
   ByVal outputFile As String, _
   ByVal compressionType As CompressionType _
) As Boolean
```

[C#]

```
public bool ExpandFile(
   string inputFile,
   string outputFile,
   CompressionType compressionType
);
```

Parameters

inputFile

A string which specifies the name of the file to be compressed. The file must exist, and it must be a regular file that can be opened for reading by the current process. An error will be returned if a character device, such as CON: is specified as the file name.

outputFile

The name of the file that is to contain the expanded file data. If the file exists, it must be a regular file that can be opened for writing by the current process and will be overwritten. If the file does not exist, it will be created. An error will be returned if a character device, such as CON: is specified as the file name.

compressionType

A CompressionType enumeration value which specifies the type of compression that was used to create the compressed data file.

Return Value

A boolean value which specifies if the method was successful or not. A return value of True indicates that the method call completed successfully. A return value of False indicates that the method failed and the application should check the value of the **LastError** property.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.ExpandFile Overload List | CompressFile | EncodeFile | EncryptFile

FileEncoder.Initialize Method

Initializes the component with the default runtime license key.

Overload List

Initializes the component with the default runtime license key.

public bool Initialize();

Initializes the component with the specified runtime license key.

public bool Initialize(string);

See Also

FileEncoder Class | SocketTools Namespace | RuntimeLicenseAttribute Class | Uninitialize Method

FileEncoder.Initialize Method ()

Initializes the component with the default runtime license key.

[Visual Basic]

```
Overloads Public Function Initialize() As Boolean
```

[C#]

```
public bool Initialize();
```

Return Value

A boolean value which specifies if the component was initialized successfully with the default runtime license key. A return value of False indicates that the license key is not valid.

Remarks

The Initialize method is used to explicitly initialize the component with the runtime license key that has been specified using the RuntimeLicense attribute. Normally it is not necessary to call this method because the component will automatically be initialized when the class constructor is called. This method should only be called if the **Uninitialize** method was previously called.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use SocketTools components.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

FileEncoder.Initialize Method (String)

Initializes the component with the specified runtime license key.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Parameters

licenseKey

A string which specifies the runtime license key.

Return Value

A boolean value which specifies if the component was initialized successfully with the runtime license key. A return value of False indicates that the license key is not valid.

Remarks

The Initialize method is used to explicitly initialize the component with a runtime license key, if one has not been specified using the RuntimeLicense attribute.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use SocketTools components.

See Also

FileEncoder Class | SocketTools Namespace | FileEncoder.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

FileEncoder.Uninitialize Method

Uninitialize the component.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The Uninitialize method explicitly releases resources allocated by the component. Normally it is not required that the application call this method since it is automatically called by the class destructor. This method should only be used if the **Initialize** method was explicitly called.

See Also

FileEncoder Class | SocketTools Namespace | Initialize Method | Reset Method

FileEncoder Events

The events of the **FileEncoder** class are listed below. For a complete list of **FileEncoder** class members, see the FileEncoder Members topic.

Public Instance Events

OnError	Occurs when a method fails.

See Also

FileEncoder Class | SocketTools Namespace

FileEncoder.OnError Event

Occurs when a method fails.

[Visual Basic]
```
Public Event OnError As OnErrorEventHandler
```

[C#]
```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type FileEncoder.ErrorEventArgs containing data related to this event. The following **FileEncoder.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Returns a description of the last error that occurred.
Error	Returns the value of the last error that has occurred.

Remarks

The event handler receives an argument of type ErrorEventArgs containing data related to this event. The following ErrorEventArgs properties provide information specific to this event.

Property	**Description**
Error	Returns the value of the last error that has occurred
Description	Returns a description of the last error that occurred

See Also

FileEncoder Class | SocketTools Namespace

FileEncoder.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see FileEncoder.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FileEncoder.ErrorEventArgs

[Visual Basic]

```
Public Class FileEncoder.ErrorEventArgs
    Inherits EventArgs
```

[C#]

```
public class FileEncoder.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileEncoder (in SocketTools.FileEncoder.dll)

See Also

FileEncoder.ErrorEventArgs Members | SocketTools Namespace

FileEncoder.ErrorEventArgs Members

FileEncoder.ErrorEventArgs overview

Public Instance Constructors

FileEncoder.ErrorEventArgs Constructor	Initializes a new instance of the FileEncoder.ErrorEventArgs class.

Public Instance Properties

Description	Returns a description of the last error that occurred.
Error	Returns the value of the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FileEncoder.ErrorEventArgs Class | SocketTools Namespace

FileEncoder.ErrorEventArgs Constructor

Initializes a new instance of the FileEncoder.ErrorEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FileEncoder.ErrorEventArgs();
```

See Also

FileEncoder.ErrorEventArgs Class | SocketTools Namespace

FileEncoder.ErrorEventArgs Properties

The properties of the **FileEncoder.ErrorEventArgs** class are listed below. For a complete list of **FileEncoder.ErrorEventArgs** class members, see the FileEncoder.ErrorEventArgs Members topic.

Public Instance Properties

Description	Returns a description of the last error that occurred.
Error	Returns the value of the last error that has occurred.

See Also

FileEncoder.ErrorEventArgs Class | SocketTools Namespace

FileEncoder.ErrorEventArgs.Description Property

Returns a description of the last error that occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the error.

See Also

FileEncoder.ErrorEventArgs Class | SocketTools Namespace

FileEncoder.ErrorEventArgs.Error Property

Returns the value of the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public FileEncoder.ErrorCode Error {get;}
```

Property Value

An ErrorCode enumeration which specifies the error.

See Also

FileEncoder.ErrorEventArgs Class | SocketTools Namespace

FileEncoder.CompressionLevel Enumeration

Provides constant values for the CompressFile and ExpandFile methods.

[Visual Basic]

```
Public Enum FileEncoder.CompressionLevel
```

[C#]

```
public enum FileEncoder.CompressionLevel
```

Members

Member Name	Description
compressionLevelDefault	The default compression level, which provides the best balance between resource usage and compression rate.
compressionLevelMinimum	The minimum compression level. This level uses the least amount of memory at the expense of a lower compression rate. This compression level requires a minimum of 8 megabytes of memory.
compressionLevel1	Compression level one. This is the minimum compression level and requires a minimum of 8 megabytes memory.
compressionLevel2	Compression level two.
compressionLevel3	Compression level three.
compressionLevel4	Compression level four.
compressionLevel5	Compression level five.
compressionLevel6	Compression level six. This is the default compression level, which provides the best balance between resource usage and compression rate.
compressionLevel7	Compression level seven.
compressionLevel8	Compression level eight.
compresisonLevel9	Compression level 9. This is the maximum compression level and requires a minimum of 32 megabytes of memory.
compressionLevelMaximum	The maximum compression level. This level uses more memory to achieve a higher compression rate. This compression level requires a minimum of 32 megabytes of memory.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileEncoder (in SocketTools.FileEncoder.dll)

See Also

SocketTools Namespace

FileEncoder.CompressionType Enumeration

Provides constant values for the CompressFile and ExpandFile methods.

[Visual Basic]
```
Public Enum FileEncoder.CompressionType
```

[C#]
```
public enum FileEncoder.CompressionType
```

Members

Member Name	Description
compressionTypeUnknown	The compression type is unknown or the data has not been compressed.
compressionTypeDefault	The default compression type. This is the same as specifying **compressionTypeDeflate**.
compressionTypeDeflate	A compression algorithm that combines LZ77 algorithm for creating common substrings and Huffman coding to process the different frequencies of byte sequences in the data stream. Deflate is widely used by compression software.
compressionTypeBurrowsWheeler	A compression algorithm that rearranges blocks of data in sorted order and then uses Huffman coding to process different frequencies of data within the block. Burrows-Wheeler compression provides a better compression ratio than the Deflate algorithm, however it requires more resources to perform the compression.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileEncoder (in SocketTools.FileEncoder.dll)

See Also

SocketTools Namespace

FileEncoder.EncodingType Enumeration

Provides constant values for the DecodeFile and EncodeFile methods.

[Visual Basic]

```
Public Enum FileEncoder.EncodingType
```

[C#]

```
public enum FileEncoder.EncodingType
```

Remarks

The EncodingType enumeration is used to specify the default encoding method used by the DecodeFile and EncodeFile methods.

Members

Member Name	Description
encodingDefault	Use the default encoding method. This is the same specifying that the base64 algorithm should be used for encoding and decoding files.
encodingBase64	Use the base64 algorithm for encoding and decoding files. This is the standard method for encoding files as outlined in the Multipurpose Internet Mail Extensions (MIME) protocol. This is the method used by most modern email client software.
encodingQuotedPrintable	This encoding method is typically used for text messages that use characters beyond the standard ASCII character set, in the range of 128-255. This method, called quoted printable encoding, allows text messages to pass through mail systems that do not support characters with the high-bit set. Note that this method should not be used to encode binary files such as executable programs because the resulting output can be very large. For binary files, use the base64 algorithm instead.
encodingUucode	Use the uuencode and uudecode algorithms for encoding and decoding files. This is a common encoding method used with UNIX systems and older email client software.
encodingYencode	Use the yEnc algorithm for encoding the file. This is an encoding method that is commonly used when posting files to Usenet newsgroups.
encodingCompressed	This option specifies the data should be compressed prior to being encoded when the EncodeFile method is called. If the DecodeFile method is called to decode a previously encoded file that was created using this option, it will cause the data to be expanded after it has been

	decoded.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileEncoder (in SocketTools.FileEncoder.dll)

See Also

SocketTools Namespace

FileEncoder.ErrorCode Enumeration

Provides constant values for error codes.

[Visual Basic]

```
Public Enum FileEncoder.ErrorCode
```

[C#]

```
public enum FileEncoder.ErrorCode
```

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.
errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.

errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorInvalidEncodingType	Invalid encoding type specified.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileEncoder (in SocketTools.FileEncoder.dll)

See Also

SocketTools Namespace

FileEncoder.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub FileEncoder.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void FileEncoder.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ErrorEventArgs that contains the event data.

Remarks

The event handler receives an argument of type ErrorEventArgs containing data related to this event. The following KeyPressEventArgs properties provide information specific to this event.

Property	Description
Error	Returns the value of the last error that has occurred
Description	Returns a description of the last error that occurred

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileEncoder (in SocketTools.FileEncoder.dll)

See Also

SocketTools Namespace

FileEncoder.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see FileEncoder.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.FileEncoder.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class FileEncoder.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class FileEncoder.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use SocketTools components.

Example

```
<Assembly: SocketTools.FileEncoder.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.FileEncoder.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileEncoder (in SocketTools.FileEncoder.dll)

See Also

FileEncoder.RuntimeLicenseAttribute Members | SocketTools Namespace

FileEncoder.RuntimeLicenseAttribute Members

FileEncoder.RuntimeLicenseAttribute overview

Public Instance Constructors

FileEncoder.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FileEncoder.RuntimeLicenseAttribute Class | SocketTools Namespace

FileEncoder.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public FileEncoder.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use SocketTools components.

See Also

FileEncoder.RuntimeLicenseAttribute Class | SocketTools Namespace

FileEncoder.RuntimeLicenseAttribute Properties

The properties of the **FileEncoder.RuntimeLicenseAttribute** class are listed below. For a complete list of **FileEncoder.RuntimeLicenseAttribute** class members, see the FileEncoder.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

FileEncoder.RuntimeLicenseAttribute Class | SocketTools Namespace

FileEncoder.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

FileEncoder.RuntimeLicenseAttribute Class | SocketTools Namespace

FileEncoderException Class

This exception is thrown when an error occurs.

For a list of all members of this type, see FileEncoderException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.FileEncoderException

[Visual Basic]
```
Public Class FileEncoderException
    Inherits ApplicationException
```

[C#]
```
public class FileEncoderException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

This exception can be thrown when a property is set that results in an error, or as the result of calling a method that fails when the **ThrowError** property has been set to True.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileEncoder (in SocketTools.FileEncoder.dll)

See Also

FileEncoderException Members | SocketTools Namespace

FileEncoderException Members

FileEncoderException overview

Public Instance Constructors

FileEncoderException	Overloaded. Initializes a new instance of the FileEncoderException class.

Public Instance Properties

HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Returns a message that describes the current exception.
Number	Return the last error that occurred.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FileEncoderException Class | SocketTools Namespace

FileEncoderException Constructor

Initializes a new instance of the FileEncoderException class without a message.

Overload List

Initializes a new instance of the FileEncoderException class without a message.

public FileEncoderException();

Initializes a new instance of the FileEncoderException class with the specific error code value.

public FileEncoderException(ErrorCode);

Initializes a new instance of the FileEncoderException class with the specific error code value.

public FileEncoderException(int);

Initializes a new instance of the FileEncoderException class with a message.

public FileEncoderException(string);

Initializes a new instance of the FileEncoderException class with a message and a reference to the inner exception that is the cause of this exception.

public FileEncoderException(string,Exception);

See Also

FileEncoderException Class | SocketTools Namespace

FileEncoderException Constructor ()

Initializes a new instance of the FileEncoderException class without a message.

[Visual Basic]

```
Overloads Public Sub New()
```

[C#]

```
public FileEncoderException();
```

See Also

FileEncoderException Class | SocketTools Namespace | FileEncoderException Constructor Overload List

FileEncoderException Constructor (String)

Initializes a new instance of the FileEncoderException class with a message.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]

```
public FileEncoderException(
   string message
);
```

Parameters

message

The message to display with this exception.

See Also

FileEncoderException Class | SocketTools Namespace | FileEncoderException Constructor Overload List

FileEncoderException Constructor (String, Exception)

Initializes a new instance of the FileEncoderException class with a message and a reference to the inner exception that is the cause of this exception.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal inner As Exception _
)
```

[C#]

```
public FileEncoderException(
   string message,
   Exception inner
);
```

Parameters

message

The message to display with this exception.

inner

The exception that is the cause of the current exception. If the inner parameter is not a null reference (Nothing in Visual Basic), the current exception is raised in a catch block that handles the inner exception.

See Also

FileEncoderException Class | SocketTools Namespace | FileEncoderException Constructor Overload List

FileEncoderException Constructor (Int32)

Initializes a new instance of the FileEncoderException class with the specific error code value.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

[C#]

```
public FileEncoderException(
   int code
);
```

Parameters

code
: An integer value which specifies the error that caused the exception to be raised.

See Also

FileEncoderException Class | SocketTools Namespace | FileEncoderException Constructor Overload List

FileEncoderException Constructor (ErrorCode)

Initializes a new instance of the FileEncoderException class with the specific error code value.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal code As ErrorCode _
)
```

[C#]

```
public FileEncoderException(
   ErrorCode code
);
```

Parameters

code

An ErrorCode enumeration which specifies the error that caused the exception to be raised.

See Also

FileEncoderException Class | SocketTools Namespace | FileEncoderException Constructor Overload List

FileEncoderException Properties

The properties of the **FileEncoderException** class are listed below. For a complete list of **FileEncoderException** class members, see the FileEncoderException Members topic.

Public Instance Properties

HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Returns a message that describes the current exception.
Number	Return the last error that occurred.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

FileEncoderException Class | SocketTools Namespace

FileEncoderException.Message Property

Returns a message that describes the current exception.

[Visual Basic]

```
Overrides Public ReadOnly Property Message As String
```

[C#]

```
public override string Message {get;}
```

Property Value

A string that explains the reason for the exception, or an empty string.

Remarks

The **Message** property returns a description of the last error that caused the exception. This can be used to display a meaningful error message to a user, rather than just the numeric value returned by the **Number** property.

See Also

FileEncoderException Class | SocketTools Namespace

FileEncoderException.Number Property

Return the last error that occurred.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An ErrorCode enumeration type that specifies the error that caused the exception.

See Also

FileEncoderException Class | SocketTools Namespace

FileEncoderException Methods

The methods of the **FileEncoderException** class are listed below. For a complete list of **FileEncoderException** class members, see the FileEncoderException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FileEncoderException Class | SocketTools Namespace

FileEncoderException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]
```
Overrides Public Function ToString() As String
```

[C#]
```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

ToString returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by ToString is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

The default implementation of ToString obtains the description of the error that caused the exception. If there is no error message or if it is an empty string (""), then no error message is returned.

This method overrides Object.ToString.

See Also

FileEncoderException Class | SocketTools Namespace

FileTransfer Class

Implements the File Transfer Protocol and Hypertext Transfer Protocol with a unified client interface.

For a list of all members of this type, see FileTransfer Members.

System.Object
  SocketTools.FileTransfer

[Visual Basic]
```
Public Class FileTransfer
    Implements IDisposable
```

[C#]
```
public class FileTransfer : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The FileTransfer class provides a comprehensive interface to the File Transfer Protocol which supports both high level operations, such as uploading or downloading files, as well as a collection of lower-level file I/O functions. In addition to file transfers, an application can create, rename and delete files and directories, search for files using wildcards and perform other common file management functions. The class library has three distinct groups of functionality:

File Transfer
Methods which enable an application to upload and download files, as well as send and receive file data using memory buffers. This gives your program the flexibility of handling the data either on disk or in memory, depending on the best needs of your application. If your program needs to transfer more than one file at a time, there are also methods which will automatically download or upload multiple files in a single method call.

File Management
In addition to transferring files, the class can be used to manage files on the server. Methods are provided to delete, rename and move files between directories. For servers that support specific protocol extensions, advanced features such as getting or setting a remote file's modification time or access permissions are also supported. If a server supports site-specific commands, such as the ability to submit a file as job on the server, the control supports this by enabling you to send custom commands to the server and then process the information that it returns.

Directory Management
The class can be used to manage directories as well as files on the server. The application can open a directory and return a list of the files that it contains, as well as create new directories and delete empty ones. The class understands a number of different directory listing formats, including those typically used on UNIX and Linux based systems, Windows server platforms, NetWare servers and VMS systems.

This class supports secure file transfers using TLS 1.2 and the SSH 2.0 protocol using SFTP.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

FileTransfer Members | SocketTools Namespace

FileTransfer Members

FileTransfer overview

Public Static (Shared) Fields

filePortFTP	A constant value which specifies the default port number for the File Transfer Protocol.
filePortFTPS	A constant value which specifies the default port number for a secure FTP connection using the SSL or TLS protocols.
filePortHTTP	A constant value which specifies the default port number for the Hypertext Transfer protocol.
filePortHTTPS	A constant value which specifies the default port number for a secure HTTP connection using the SSL or TLS protocols.
filePortSFTP	A constant value which specifies the default port number for a secure SSH connection using the SSH1 or SSH2 protocols.
fileTimeout	A constant value which specifies the default timeout period.

Public Static (Shared) Methods

ErrorText	Returns the description of an error code.

Public Instance Constructors

FileTransfer Constructor	Initializes a new instance of the FileTransfer class.

Public Instance Properties

Account	Get or sets a value that specifies the account name for the current user.
ActivePorts	Gets and sets the port numbers used for active mode file transfers.
CertificateExpires	Get a value that specifies the date that the security certificate expires.
CertificateIssued	Get a value that specifies the date that the security certificate was issued.
CertificateIssuer	Get a value that provides information about the organization that issued the certificate.
CertificateName	Gets and sets a value that specifies the name of the client certificate.
CertificatePassword	Gets and sets the password associated with the client certificate.
CertificateStatus	Gets a value which indicates the status of the security certificate returned by the server.

CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateSubject	Gets a value that provides information about the organization that the server certificate was issued to.
CertificateUser	Gets and sets the user that owns the client certificate.
ChannelMode	Set or return the security mode for the specified communications channel.
CipherStrength	Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.
Compression	Gets and sets a value that specifies if data compression should be enabled.
DirectoryFormat	Gets and sets a value which specifies the current directory format type.
Encoding	Gets and sets the character encoding that is used when a file name is sent to the server.
Features	Gets and sets the features that are currently enabled for the current session.
FileType	Gets and sets a value which specifies the type of file that is being transferred.
Handle	Gets a value that specifies the client handle allocated for the current session.
HashStrength	Gets a value which specifies the length of the message digest that was selected for a secure connection.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsConnected	Gets a value which indicates if a connection to the server has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
KeepAlive	Gets and sets a Boolean value which specifies if the client should attempt to keep the server connection alive.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets the local Internet address that the client is bound to.
LocalFile	Gets and sets the name of the file on the local

	system.
Localize	Gets and sets a value which specifies if time and dates should be adjusted for the current timezone.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
Options	Gets and sets a value which specifies one or more client options.
Passive	Gets and sets a value which specifies if passive mode file transfers should be enabled.
Password	Gets and sets the password used to authenticate the client session.
Priority	Gets and sets a value which specifies the priority of file transfers.
ProtocolVersion	Gets and sets a value which specifies the default protocol version.
ProxyPassword	Gets and sets the password used to authenticate the connection to a proxy server.
ProxyPort	Gets and sets a value that specifies the proxy server port number.
ProxyServer	Gets and sets the hostname or IP address of a proxy server.
ProxyType	Gets and sets the type of proxy server the client will use to establish a connection.
ProxyUser	Gets and sets the username used to authenticate the connection to a proxy server.
RemoteFile	Gets and sets a value which specifies a file name on the server.
ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.
Secure	Gets and sets a value which specifies if a secure connection is established.
SecureCipher	Gets a value that specifies the encryption algorithm used for a secure connection.
SecureHash	Gets a value that specifies the message digest algorithm used for a secure connection.
SecureKeyExchange	Gets a value that specifies the key exchange algorithm used for a secure connection.
SecureProtocol	Gets and sets a value which specifies the protocol used for a secure connection.

ServerDirectory	Gets and sets a value which specifies the current working directory on the file server.
ServerName	Gets and sets a value which specifies the host name used to establish a connection.
ServerPort	Gets and sets a value which specifies the remote port number.
ServerType	Gets and sets a value which specifies the type of file server the client is connecting to.
System	Gets a string value which identifies the server.
TaskCount	Get the number of active background file transfers.
TaskId	Get the task identifier for the last background file transfer.
TaskList	Get an array of active background task identifiers.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
TimeZone	Gets and sets the current timezone offset in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
TransferBytes	Gets a value which specifies the number of bytes transferred to or from the remote server.
TransferRate	Gets a value which specifies the data transfer rate in bytes per second.
TransferTime	Gets a value which specifies the number of seconds elapsed during a data transfer.
URL	Gets and sets the current URL used to access a file on the server.
UserName	Gets and sets the username used to authenticate the client session.
Version	Gets a value which returns the current version of the FileTransfer class library.

Public Instance Methods

AddFileType	Associate a file name extension with a specific file type.
AsyncGetFile	Overloaded. Download a file from the server to the local system in the background.
AsyncPutFile	Overloaded. Upload a file from the local system to the server in the background.
AttachThread	Attach an instance of the class to the current thread
Cancel	Cancel the current blocking client operation.
ChangeDirectory	Change the current working directory on the remote server.
CloseDirectory	Close the directory that was previously opened with the OpenDirectory method.
Command	Overloaded. Send a custom command to the server.
Connect	Overloaded. Establish a connection with a file server.
DeleteFile	Delete a file on the remote server.
Disconnect	Terminate the connection with the remote server.
Dispose	Overloaded. Releases all resources used by FileTransfer.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetData	Overloaded. Transfers the contents of a file on the server and stores it in byte array.
GetDirectory	Return the current working directory.
GetFile	Overloaded. Download a file from the server to the local system.
GetFileList	Overloaded. Returns an unparsed list of files in the specified directory.
GetFilePermissions	Overloaded. Return the access permissions for a file on the remote system.
GetFileSize	Overloaded. Returns the size of the specified file on the remote server.
GetFileStatus	
GetFileTime	Overloaded. Returns the modification date and time for specified file on the remote server.
GetFirstFile	Overloaded. Get information about the first file in a directory listing returned by the server.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.

GetMultipleFiles	Download multiple files from the server to the local system using a wildcard mask.
GetNextFile	Overloaded. Get information about the next file in a directory listing returned by the server.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the FileTransfer class.
Login	Overloaded. Login to the remote server.
Logout	Log the current user off the server.
MakeDirectory	Create a new directory on the server.
OpenDirectory	Overloaded. Open the specified directory on the server.
PostFile	Overloaded. Post the contents of the specified file to a script executed on the remote server.
PutData	Overloaded. Transfers data from a byte array and stores it in a file on the remote server.
PutFile	Overloaded. Upload a file from the local system to the server.
PutMultipleFiles	Upload multiple files from the local system to the server using a wildcard mask.
RemoveDirectory	Remove a directory on the server.
RenameFile	Change the name of a file on the server.
Reset	Reset the internal state of the object, resetting all properties to their default values.
SetFilePermissions	Change the access permissions for a file on the server.
SetFileTime	Changes the modification date and time for a file on the server.
TaskAbort	Overloaded. Abort the specified asynchronous task.
TaskDone	Overloaded. Determine if an asynchronous task has completed.
TaskResume	Overloaded. Resume execution of an asynchronous task.
TaskSuspend	Overloaded. Suspend execution of an asynchronous task.
TaskWait	Overloaded. Wait for an asynchronous task to complete.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.

VerifyFile	Overloaded. Verify that the contents of a file on the local system are the same as the specified file on the server.

Public Instance Events

OnCancel	Occurs when a blocking client operation is canceled.
OnCommand	Occurs when the client sends a command to the server and receives a reply indicating the result of that command.
OnError	Occurs when an network operation fails.
OnGetFile	Occurs when a file download has been initiated.
OnProgress	Occurs as a data stream is being read or written to the server.
OnPutFile	Occurs when a file upload is initiated.
OnTaskBegin	Occurs when an asynchronous task begins execution.
OnTaskEnd	Occurs when an asynchronous task completes.
OnTaskRun	Occurs while a background task is active.
OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the FileTransfer class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer Constructor

Initializes a new instance of the FileTransfer class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FileTransfer();
```

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer Fields

The fields of the **FileTransfer** class are listed below. For a complete list of **FileTransfer** class members, see the FileTransfer Members topic.

Public Static (Shared) Fields

filePortFTP	A constant value which specifies the default port number for the File Transfer Protocol.
filePortFTPS	A constant value which specifies the default port number for a secure FTP connection using the SSL or TLS protocols.
filePortHTTP	A constant value which specifies the default port number for the Hypertext Transfer protocol.
filePortHTTPS	A constant value which specifies the default port number for a secure HTTP connection using the SSL or TLS protocols.
filePortSFTP	A constant value which specifies the default port number for a secure SSH connection using the SSH1 or SSH2 protocols.
fileTimeout	A constant value which specifies the default timeout period.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.filePortFTP Field

A constant value which specifies the default port number for the File Transfer Protocol.

[Visual Basic]

```
Public Const filePortFTP As Integer = 21
```

[C#]

```
public const int filePortFTP = 21;
```

Remarks

The default port number for the File Transfer Protocol is 21.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.filePortFTPS Field

A constant value which specifies the default port number for a secure FTP connection using the SSL or TLS protocols.

[Visual Basic]

```
Public Const filePortFTPS As Integer = 990
```

[C#]

```
public const int filePortFTPS = 990;
```

Remarks

The default port number for a secure connection using SSL or TLS is 990.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.filePortHTTP Field

A constant value which specifies the default port number for the Hypertext Transfer protocol.

[Visual Basic]

```
Public Const filePortHTTP As Integer = 80
```

[C#]

```
public const int filePortHTTP = 80;
```

Remarks

The default port number for the Hypertext Transfer Protocol is 80.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.filePortHTTPS Field

A constant value which specifies the default port number for a secure HTTP connection using the SSL or TLS protocols.

[Visual Basic]

```
Public Const filePortHTTPS As Integer = 443
```

[C#]

```
public const int filePortHTTPS = 443;
```

Remarks

The default port number for a secure connection using SSL or TLS is 443.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.filePortSFTP Field

A constant value which specifies the default port number for a secure SSH connection using the SSH1 or SSH2 protocols.

[Visual Basic]
```
Public Const filePortSFTP As Integer = 22
```

[C#]
```
public const int filePortSFTP = 22;
```

Remarks

The default port number for a secure connection using SSH is 22.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.fileTimeout Field

A constant value which specifies the default timeout period.

[Visual Basic]

```
Public Const fileTimeout As Integer = 20
```

[C#]

```
public const int fileTimeout = 20;
```

Remarks

The default timeout period is 20 seconds for each blocking network operation. An error will occur if the operation does not complete within the specified time period.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer Properties

The properties of the **FileTransfer** class are listed below. For a complete list of **FileTransfer** class members, see the FileTransfer Members topic.

Public Instance Properties

Account	Get or sets a value that specifies the account name for the current user.
ActivePorts	Gets and sets the port numbers used for active mode file transfers.
CertificateExpires	Get a value that specifies the date that the security certificate expires.
CertificateIssued	Get a value that specifies the date that the security certificate was issued.
CertificateIssuer	Get a value that provides information about the organization that issued the certificate.
CertificateName	Gets and sets a value that specifies the name of the client certificate.
CertificatePassword	Gets and sets the password associated with the client certificate.
CertificateStatus	Gets a value which indicates the status of the security certificate returned by the server.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateSubject	Gets a value that provides information about the organization that the server certificate was issued to.
CertificateUser	Gets and sets the user that owns the client certificate.
ChannelMode	Set or return the security mode for the specified communications channel.
CipherStrength	Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.
Compression	Gets and sets a value that specifies if data compression should be enabled.
DirectoryFormat	Gets and sets a value which specifies the current directory format type.
Encoding	Gets and sets the character encoding that is used when a file name is sent to the server.
Features	Gets and sets the features that are currently enabled for the current session.
FileType	Gets and sets a value which specifies the type of

	file that is being transferred.
Handle	Gets a value that specifies the client handle allocated for the current session.
HashStrength	Gets a value which specifies the length of the message digest that was selected for a secure connection.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsConnected	Gets a value which indicates if a connection to the server has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
KeepAlive	Gets and sets a Boolean value which specifies if the client should attempt to keep the server connection alive.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets the local Internet address that the client is bound to.
LocalFile	Gets and sets the name of the file on the local system.
Localize	Gets and sets a value which specifies if time and dates should be adjusted for the current timezone.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
Options	Gets and sets a value which specifies one or more client options.
Passive	Gets and sets a value which specifies if passive mode file transfers should be enabled.
Password	Gets and sets the password used to authenticate the client session.
Priority	Gets and sets a value which specifies the priority of file transfers.
ProtocolVersion	Gets and sets a value which specifies the default protocol version.
ProxyPassword	Gets and sets the password used to authenticate the connection to a proxy server.
ProxyPort	Gets and sets a value that specifies the proxy server port number.

ProxyServer	Gets and sets the hostname or IP address of a proxy server.
ProxyType	Gets and sets the type of proxy server the client will use to establish a connection.
ProxyUser	Gets and sets the username used to authenticate the connection to a proxy server.
RemoteFile	Gets and sets a value which specifies a file name on the server.
ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.
Secure	Gets and sets a value which specifies if a secure connection is established.
SecureCipher	Gets a value that specifies the encryption algorithm used for a secure connection.
SecureHash	Gets a value that specifies the message digest algorithm used for a secure connection.
SecureKeyExchange	Gets a value that specifies the key exchange algorithm used for a secure connection.
SecureProtocol	Gets and sets a value which specifies the protocol used for a secure connection.
ServerDirectory	Gets and sets a value which specifies the current working directory on the file server.
ServerName	Gets and sets a value which specifies the host name used to establish a connection.
ServerPort	Gets and sets a value which specifies the remote port number.
ServerType	Gets and sets a value which specifies the type of file server the client is connecting to.
System	Gets a string value which identifies the server.
TaskCount	Get the number of active background file transfers.
TaskId	Get the task identifier for the last background file transfer.
TaskList	Get an array of active background task identifiers.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.

TimeZone	Gets and sets the current timezone offset in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
TransferBytes	Gets a value which specifies the number of bytes transferred to or from the remote server.
TransferRate	Gets a value which specifies the data transfer rate in bytes per second.
TransferTime	Gets a value which specifies the number of seconds elapsed during a data transfer.
URL	Gets and sets the current URL used to access a file on the server.
UserName	Gets and sets the username used to authenticate the client session.
Version	Gets a value which returns the current version of the FileTransfer class library.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.Account Property

Get or sets a value that specifies the account name for the current user.

[Visual Basic]

```
Public Property Account As String
```

[C#]

```
public string Account {get; set;}
```

Property Value

A string which specifies the account name for the current user.

Remarks

The **Account** property specifies the account name of the current user, if it is required by the server for authentication. Not all servers require an account name, in which case this property is ignored.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.ActivePorts Property

Gets and sets the port numbers used for active mode file transfers.

[Visual Basic]

```
Public Property ActivePorts As PortRange
```

[C#]

```
public FileTransfer.PortRange ActivePorts {get; set;}
```

Property Value

A **PortRange** structure which specifies the minimum and maximum port numbers used for active mode file transfers.

Remarks

This property is used to change the default port numbers used for active mode file transfers. Active mode is used when the **Passive** property is set to false. Instead of the client establishing an outbound connection to the server for the file transfer, it listens for an inbound connection from the server back to the client. In most cases, passive mode transfers are preferred because they mitigate potential compatibility issues with firewalls and NAT routers.

If active mode transfers are required, the default port range used when listening for the server connection is between 1024 and 5000. This is the standard range of ephemeral ports used by the Windows operating system. However, under some circumstances that range of ports may be too small, or a firewall may be configured to deny inbound connections on ephemeral ports. In that case, the **ActivePorts** property can be used to specify a different range of port numbers.

While it is technically permissible to assign the low and high port numbers to the same value, effectively specifying a single active port number, this is not recommended as it can cause the transfer to fail unexpectedly if multiple file transfers are performed. A minimum range of at least 1000 ports is recommended. For example, if you specify a low port value of 40000 then it is recommended that the high port value be at least 41000. The maximum port value is 65535.

See Also

FileTransfer Class | SocketTools Namespace | PortRange Structure

FileTransfer.CertificateExpires Property

Get a value that specifies the date that the security certificate expires.

[Visual Basic]

```
Public ReadOnly Property CertificateExpires As String
```

[C#]

```
public string CertificateExpires {get;}
```

Property Value

A string which specifies a date using the local date and time format.

Remarks

The **CertificateExpires** property returns a string that specifies the date and time that the security certificate expires. This property will return an empty string if a secure connection has not been established with the remote host.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.CertificateIssued Property

Get a value that specifies the date that the security certificate was issued.

[Visual Basic]
```
Public ReadOnly Property CertificateIssued As String
```

[C#]
```
public string CertificateIssued {get;}
```

Property Value

A string which specifies a date using the local date and time format.

Remarks

The **CertificateIssued** property returns a string that specifies the date and time that the security certificate was issued. This property will return an empty string if a secure connection has not been established with the remote host.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.CertificateIssuer Property

Get a value that provides information about the organization that issued the certificate.

[Visual Basic]

```
Public ReadOnly Property CertificateIssuer As String
```

[C#]

```
public string CertificateIssuer {get;}
```

Property Value

A string that contains a comma separated list of name value pairs.

Remarks

The **CertificateIssuer** property returns a string that contains information about the organization that issued the server certificate. The string value is a comma separated list of tagged name and value pairs. In the nomenclature of the X.500 standard, each of these pairs are called a relative distinguished name (RDN), and when concatenated together, forms the issuer's distinguished name (DN). For example:

```
C=US, O="RSA Data Security, Inc.", OU=Secure Server Certification Authority
```

To obtain a specific value, such as the name of the issuer or the issuer's country, the application must parse the string returned by this property. Some of the common tokens used in the distinguished name are:

Token	Description
C	The ISO standard two character country code.
S	The name of the state or province.
L	The name of the city or locality.
O	The name of the company or organization.
OU	The name of the department or organizational unit
CN	The common name; with X.509 certificates, this is the domain name of the site the certificate was issued for.

This property will return an empty string if a secure connection has not been established with the server.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.CertificateName Property

Gets and sets a value that specifies the name of the client certificate.

[Visual Basic]
```
Public Property CertificateName As String
```

[C#]
```
public string CertificateName {get; set;}
```

Property Value

A string which specifies the certificate name.

Remarks

The **CertificateName** property is used to specify the name of a client certificate to use when establishing a secure connection. It is only required that you set this property value if the server requires a client certificate for authentication. If this property is not set, a client certificate will not be provided to the server. If a certificate name is specified, the certificate must have a private key associated with it, otherwise the connection attempt will fail because the control will be unable to create a security context for the session.

When the certificate store is searched for a matching certificate, it will first search for any certificate with a friendly name that matches the property value. If no valid certificate is found, it will then search for a certificate with a matching common name.

Certificates may be installed and viewed on the local system using the Certificate Manager that is included with the Windows operating system. For more information, refer to the documentation for the Microsoft Management Console.

See Also

FileTransfer Class | SocketTools Namespace | CertificateStore Property | Secure Property

FileTransfer.CertificateStatus Property

Gets a value which indicates the status of the security certificate returned by the server.

[Visual Basic]

```
Public ReadOnly Property CertificateStatus As SecurityCertificate
```

[C#]

```
public FileTransfer.SecurityCertificate CertificateStatus {get;}
```

Property Value

A SecurityCertificate enumeration value which specifies the status of the certificate.

Remarks

The **CertificateStatus** property is used to determine the status of the security certificate returned by the server when a secure connection has been established. This property value should be checked after the connection to the server has completed, but prior to beginning a transaction.

Note that if the certificate cannot be validated, the secure connection will not be automatically terminated. It is the responsibility of your application to determine the best course of action to take if the certificate is invalid. Even if the security certificate cannot be validated, the data exchanged with the remote host will still be encrypted.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.CertificateStore Property

Gets and sets a value that specifies the name of the local certificate store.

[Visual Basic]

```
Public Property CertificateStore As String
```

[C#]

```
public string CertificateStore {get; set;}
```

Property Value

A string which specifies the certificate store name. The default value is the current user's personal certificate store.

Remarks

The **CertificateStore** property is used to specify the name of the certificate store which contains the security certificate to use when establishing a secure connection. The certificate may either be stored in the registry or in a file. If the certificate is stored in the registry, then this property should be set to one of the following predefined values:

Store Name	Description
CA	Certification authority certificates. These are certificates that are issued by entities which are entrusted to issue certificates to other individuals or organizations. Companies such as VeriSign and Thawte act as certification authorities.
MY	Personal certificates and their associated private keys for the current user. This store typically holds the client certificates used to establish a user's credentials. If a certificate store is not specified, this is the default value that is used.
ROOT	Certificates that have been self-signed by a certificate authority. Root certificates for a number of different certification authorities such as VeriSign and Thawte are installed as part of the operating system and periodically updated by Microsoft.

In most cases the client certificate will be installed in the user's personal certificate store, and therefore it is not necessary to set this property value because that is the default location that will be used to search for the certificate. This property is only used if the **CertificateName** property is also set to a valid certificate name.

If you are using a local certificate store, with the certificate and private key stored in the registry, you can explicitly specify whether the certificate store for the current user or the local machine (all users) should be used. This is done by prefixing the certificate store name with "HKCU" for the current user, or "HKLM" for the local machine. For example, a certificate store name of "HKLM:MY" would specify the personal certificate store for the local machine, rather than the current user. If neither prefix is specified, it will default to the certificate store for the current user.

This property may also be used to specify a file that contains the client certificate. In this case, the property should specify the full path to the file and must contain both the certificate and private key in PKCS12

format. If the file is protected by a password, the **CertificatePassword** property must also be set to specify the password.

See Also

FileTransfer Class | SocketTools Namespace | CertificatePassword Property | Secure Property

FileTransfer.CertificateSubject Property

Gets a value that provides information about the organization that the server certificate was issued to.

[Visual Basic]
```
Public ReadOnly Property CertificateSubject As String
```

[C#]
```
public string CertificateSubject {get;}
```

Property Value

A string that contains a comma separated list of name value pairs.

Remarks

The **CertificateSubject** property returns a string that contains information about the organization that the server certificate was issued to. The string value is a comma separated list of tagged name and value pairs. In the nomenclature of the X.500 standard, each of these pairs are called a relative distinguished name (RDN), and when concatenated together, forms the issuer's distinguished name (DN). For example:

```
C=US, O="RSA Data Security, Inc.", OU=Secure Server Certification Authority
```

To obtain a specific value, such as the name of the issuer or the issuer's country, the application must parse the string returned by this property. Some of the common tokens used in the distinguished name are:

Token	Description
C	The ISO standard two character country code.
S	The name of the state or province.
L	The name of the city or locality.
O	The name of the company or organization.
OU	The name of the department or organizational unit
CN	The common name; with X.509 certificates, this is the domain name of the site the certificate was issued for.

This property will return an empty string if a secure connection has not been established with the server.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.ChannelMode Property

Set or return the security mode for the specified communications channel.

[Visual Basic]

```
Public Property ChannelMode As SecureChannel
```

[C#]

```
public FileTransfer.SecureChannel ChannelMode {get; set;}
```

Property Value

An FtpChannelMode enumeration value which specifies the current channel mode.

Remarks

The **ChannelMode** property is used to change the default mode for the specified channel, and is typically used to control whether or not data is encrypted during a file transfer. If a standard, non-secure connection has been established with the server, an error will be returned if you specify the **channelSecure** mode for either channel.

If you have established a secure connection and then specify the **channelClear** mode for the command channel, the client will send the CCC command to the server to indicate that commands should no longer be encrypted. If the server does not support this command, an error will be returned and the channel mode will remain unchanged. Once the command channel has been changed to clear mode, it cannot be changed back to secure mode. You must disconnect and re-connect to the server if you want to resume sending commands over an encrypted channel.

Changing the mode for the data channel requires that the server support the PROT command. If this command is not supported by the server, an exception will be thrown which must be handled by the application. You can only set a channel to secure mode if the **Secure** property is also set to **true**.

It is important to note that this property should only be used after a connection has been established with the server. If you attempt to read the property or change a value prior to calling the **Connect** method, an exception will be thrown.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.CipherStrength Property

Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.

[Visual Basic]

```
Public ReadOnly Property CipherStrength As Integer
```

[C#]

```
public int CipherStrength {get;}
```

Property Value

An integer value which specifies the encryption key length if a secure connection has been established; otherwise a value of 0 is returned.

Remarks

The **CipherStrength** property returns the number of bits in the key used to encrypt the secure data stream. Common values returned by this property are 128 and 256. A key length of 40 or 56 bits is considered insecure and subject to brute force attacks. 128-bit and 256-bit keys are considered secure. If this property returns a value of 0, this means that a secure connection has not been established with the server.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.Compression Property

Gets and sets a value that specifies if data compression should be enabled.

[Visual Basic]
```
Public Property Compression As Boolean
```

[C#]
```
public bool Compression {get; set;}
```

Property Value

Returns **true** if a data compression is enabled; otherwise returns **false**. The default value is **false.**

Remarks

The **Compression** property is used to indicate to the server whether or not it is acceptable to compress the data that is returned to the client. If compression is enabled, the client will advertise that it will accept compressed data and the server will decide whether a resource being requested can be compressed. If the data is compressed, the control will automatically expand the data before returning it to the caller.

Enabling compression does not guarantee that the data returned by the server will actually be compressed, it only informs the server that the client is willing to accept compressed data. Whether or not a particular resource is compressed depends on the server configuration, and the server may decide to only compress certain types of resources, such as text files. Disabling compression informs the server that the client is not willing to accept compressed data; this is the default.

This property value is only meaningful when downloading files from an HTTP server that supports file compression. It has no effect on file uploads or when transferring files using FTP.

See Also

FileTransfer Class | SocketTools Namespace | GetData Method | GetFile Method

FileTransfer.DirectoryFormat Property

Gets and sets a value which specifies the current directory format type.

[Visual Basic]
```
Public Property DirectoryFormat As FtpDirectoryFormat
```

[C#]
```
public FileTransfer.FtpDirectoryFormat DirectoryFormat {get; set;}
```

Property Value

An FtpDirectoryFormat enumeration value which specifies the current directory format.

Remarks

This property should only be set if the client cannot automatically determine the directory format returned by the server. The default directory format is determined both by the server's operating system and by analyzing the format of the data returned by the server. If the class is unable to automatically determine the format, it will attempt to parse the list of files as though it is a UNIX style listing.

If this property is set to the default value **FtpDirectoryFormat.formatAuto** and the class can determine from the format of the file listing returned by the server, then the property will change value upon the first call to the **GetFirstFile** method or the first time the **OnFileList** event is generated.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.Features Property

Gets and sets the features that are currently enabled for the current session.

[Visual Basic]

```
Public Property Features As FtpFeatures
```

[C#]

```
public FileTransfer.FtpFeatures Features {get; set;}
```

Property Value

An FtpFeatures enumeration which specifies the features that are available for the current client session.

Remarks

When a client connection is first established, all features are enabled by default. However, as the client issues commands to the server, if the server reports that the command is unrecognized that feature will automatically be disabled in the client.

For example, the first time an application calls the **GetFileSize** method to determine the size of a file, the control will try to use the SIZE command. If the server reports that the SIZE command is not available, that feature will be disabled and the control will not use the command again during the session unless it is explicitly re-enabled. This is designed to prevent the control from repeatedly sending invalid commands to a server, which may result in the server aborting the connection.

Setting the **Features** property enables those features which have been specified. More than one feature may be enabled by combining the above constants using a bitwise Or operator. To test if a particular feature has been enabled, use the bitwise And operator.

Because features are specific to the current session, once you disconnect from the server they are reset. Even if you wish to reconnect to the same server, you must explicitly set the **Features** property again to those features which you wish to enable. Setting the **Features** property when the control is not connected to a server will cause the client session to only use those specified features for the next connection that is established. Setting the **Features** property during an active connection will change the features available for that session.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.FileType Property

Gets and sets a value which specifies the type of file that is being transferred.

[Visual Basic]

```
Public Property FileType As FileTransferType
```

[C#]

```
public FileTransfer.FileTransferType FileType {get; set;}
```

Property Value

An **FtpFileType** enumeration which specifies the type of file being uploaded or downloaded.

Remarks

The file type should be set before a file is uploaded or downloaded from the remote server. Once the file type is set, it is in effect for all files that are subsequently transferred.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.Handle Property

Gets a value that specifies the client handle allocated for the current session.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer which represents a client handle. If there is no active connection, a value of -1 is returned.

Remarks

The **Handle** property specifies the numeric descriptor of the current client session and is typically used for debugging or diagnostic purposes.

In SocketTools, handles are used to identify client sessions. A session begins when an instance of the class is used to establish a connection with the server and ends when that connection is terminated. The client handle is defined as an integer type and is used internally to reference the active session. When the connection is terminated, the handle is released, along with any system resources that were allocated for it. An unused handle is identified by the value -1.

It is important to note that the handles returned by this property are not socket handles and cannot be used interchangeably with other objects or Windows API functions. The actual value of the handle is only unique while the client session is active and handle values may be reused. An application should never depend on the **Handle** property returning a specific value.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.HashStrength Property

Gets a value which specifies the length of the message digest that was selected for a secure connection.

[Visual Basic]

```
Public ReadOnly Property HashStrength As Integer
```

[C#]

```
public int HashStrength {get;}
```

Property Value

An integer value which specifies the length of the message digest if a secure connection has been established; otherwise a value of 0 is returned.

Remarks

The **HashStrength** property returns the number of bits used in the message digest (hash) that was selected. Common values returned by this property are 128 and 160. If this property returns a value of 0, this means that a secure connection has not been established with the server.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.IsBlocked Property

Gets a value which indicates if the current thread is performing a blocking client operation.

[Visual Basic]

```
Public ReadOnly Property IsBlocked As Boolean
```

[C#]

```
public bool IsBlocked {get;}
```

Property Value

Returns **true** if the current thread is blocking, otherwise returns **false**.

Remarks

The **IsBlocked** property returns **true** if the current thread is blocked performing an operation. Because the Windows Sockets API only permits one blocking operation per thread of execution, this property should be checked before starting any blocking operation in response to an event.

If the **IsBlocked** property returns **false**, this means there are no blocking operations on the current thread at that time. However, this does not guarantee that the next client operation will not fail. An application should always check the return value from a client operation and check the value of the **LastError** property if an error occurs.

Note that this property will return **true** if there is any blocking operation being performed by the current thread, regardless of whether this specific instance of the class is responsible for the blocking operation or not.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.IsConnected Property

Gets a value which indicates if a connection to the server has been established.

[Visual Basic]

```
Public ReadOnly Property IsConnected As Boolean
```

[C#]

```
public bool IsConnected {get;}
```

Property Value

Returns **true** if the connection has been established; otherwise returns **false**.

Remarks

The **IsConnected** property can only be used to indicate if there is still a logical connection to the server. It cannot be used to detect abnormal conditions such as the server aborting the connection, the physical network connection being lost or other critical errors.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.IsInitialized Property

Gets a value which indicates if the current instance of the class has been initialized successfully.

[Visual Basic]

```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]

```
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the class instance has been initialized; otherwise returns **false**.

Remarks

The **IsInitialized** property is used to determine if the current instance of the class has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the class may not be able to initialize itself.

The most common reasons that a class instance may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required networking libraries or not being able to access the system registry.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.KeepAlive Property

Gets and sets a Boolean value which specifies if the client should attempt to keep the server connection alive.

[Visual Basic]
```
Public Property KeepAlive As Boolean
```

[C#]
```
public bool KeepAlive {get; set;}
```

Property Value

A boolean value which specifies if the client should attempt to maintain the connection with the server over a long period of time.

Remarks

If the **KeepAlive** property is set to **true**, the client will attempt to maintain an active connection to the server over a long period of time. If this property is set to **false**, then no attempt will be made to hold the command channel open.

It is important to note that enabling this option does not guarantee that the connection will be maintained. The application must be written to account for situations where the connection to the server is terminated if it is idle for a long period of time, regardless of the value of this property.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As Integer
```

[C#]

```
public int LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.LocalAddress Property

Gets the local Internet address that the client is bound to.

[Visual Basic]

```
Public ReadOnly Property LocalAddress As String
```

[C#]

```
public string LocalAddress {get;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **LocalAddress** property returns the local Internet address that the client is bound to when a connection is established with a server. This property may return either an IPv4 or IPv6 formatted address, depending on the type of connection that was established.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.LocalFile Property

Gets and sets the name of the file on the local system.

[Visual Basic]

```
Public Property LocalFile As String
```

[C#]

```
public string LocalFile {get; set;}
```

Property Value

A string value which specifies the name of the local file.

Remarks

The **LocalFile** property is used to specify the local file name that will be used when uploading or downloading files. This property is automatically updated whenever the **GetFile** or **PutFile** methods are called, specifying the local file name.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.Localize Property

Gets and sets a value which specifies if time and dates should be adjusted for the current timezone.

[Visual Basic]

```
Public Property Localize As Boolean
```

[C#]

```
public bool Localize {get; set;}
```

Property Value

A boolean value which specifies if file time and dates should be adjusted for the local timezone. The default value is false.

Remarks

The **Localize** property controls how remote file date and time values are localized when the **GetFileTime** method is called. If the property is set to **true** the file date and time will be adjusted to the current timezone. If the property is set to **false** the file date and time are returned as UTC (Coordinated Universal Time) values.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.LocalName Property

Gets a value which specifies the host name for the local system.

[Visual Basic]

```
Public ReadOnly Property LocalName As String
```

[C#]

```
public string LocalName {get;}
```

Property Value

A string which specifies the local host name.

Remarks

The **LocalName** property returns the fully-qualified host name assigned to the local system. If the system has not been configured with an Internet domain name, then this property will return the NetBIOS name assigned to the local system.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.Options Property

Gets and sets a value which specifies one or more client options.

[Visual Basic]

```
Public Property Options As FileTransferOptions
```

[C#]

```
public FileTransfer.FileTransferOptions Options {get; set;}
```

Property Value

Returns one or more FileTransferOptions enumeration flags which specify the options for the client. The default value for this property is **optionNone**.

Remarks

The **Options** property specifies one or more default options options which are used when establishing a connection using the **Connect** method.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.Passive Property

Gets and sets a value which specifies if passive mode file transfers should be enabled.

[Visual Basic]

```
Public Property Passive As Boolean
```

[C#]

```
public bool Passive {get; set;}
```

Property Value

A boolean value which specifies if passive mode file transfers are enabled. If this value is set to **true**, passive mode is enabled. If the value is set to **false**, then passive mode transfers are disabled. The default value is **true**.

Remarks

When the client uploads or downloads a file and the **Passive** property is set to **false**, the server establishes a second connection back to the client which is used to transfer the file data. However, if the local system is behind a firewall or a NAT router, the server may not be able to create the data connection and the transfer will fail. By setting this property to **true**, it forces the client to establish an outbound data connection with the server. It is recommended that most applications use passive mode whenever possible.

Setting this property to **true** is the same as specifying the optionPassive flag when establishing a connection to the server.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.Password Property

Gets and sets the password used to authenticate the client session.

[Visual Basic]

```
Public Property Password As String
```

[C#]

```
public string Password {get; set;}
```

Property Value

A string which specifies the password.

Remarks

If a password is not specified when the **Connect** method is called, the value of this property will be used as the default password when establishing a connection with the server.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.Priority Property

Gets and sets a value which specifies the priority of file transfers.

[Visual Basic]
```
Public Property Priority As FileTransferPriority
```

[C#]
```
public FileTransfer.FileTransferPriority Priority {get; set;}
```

Property Value

Returns a FileTransferPriority enumeration value which specify the current file transfer priority. The default value for this property is **priorityNormal**.

Remarks

The **Priority** property can be used to control the processor usage, memory and network bandwidth allocated for file transfers. The default priority balances resource utilization and transfer speed while ensuring that a single-threaded application remains responsive to the user. Lower priorities reduce the overall resource utilization at the expense of transfer speed. For example, if you create a worker thread to download a file in the background and want to ensure that it has a minimal impact on the process, the **priorityBackground** value can be used.

Higher priority values increase the memory allocated for the transfers and increases processor utilization for the transfer. The **priorityCritical** priority maximizes transfer speed at the expense of system resources. It is not recommended that you increase the file transfer priority unless you understand the implications of doing so and have thoroughly tested your application. If the file transfer is being performed in the main UI thread, increasing the priority may interfere with the normal processing of Windows messages and cause the application to appear to become non-responsive. It is also important to note that when the priority is set to **priorityCritical**, normal progress events will not be generated during the transfer.

See Also

FileTransfer Class | SocketTools Namespace | FileTransferPriority Enumeration

FileTransfer.ProtocolVersion Property

Gets and sets a value which specifies the default protocol version.

[Visual Basic]

```
Public Property ProtocolVersion As HttpVersion
```

[C#]

```
public FileTransfer.HttpVersion ProtocolVersion {get; set;}
```

Property Value

An HttpVersion enumeration which specifies the protocol version.

Remarks

The **ProtocolVersion** property sets or returns the current HTTP version number. It is used to determine how requests are submitted to the server, as well as what header fields are required. The default value for this property is **HttpVersion.version10**, and should be changed before any connection attempt is made by the client.

Note that setting the property value to **HttpVersion.version09** tells the client to use the preliminary protocol specification which only supported a basic version of the GET command, and did not have any provisions for features such as user authentication, virtual hosting, etc. Header fields are not supported in this version of the protocol.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.ProxyPassword Property

Gets and sets the password used to authenticate the connection to a proxy server.

[Visual Basic]

```
Public Property ProxyPassword As String
```

[C#]

```
public string ProxyPassword {get; set;}
```

Property Value

A string which specifies a password.

Remarks

The **ProxyPassword** property specifies the password used to authenticate the user to the proxy server. If a password is not required by the server, this property is ignored.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.ProxyPort Property

Gets and sets a value that specifies the proxy server port number.

[Visual Basic]

```
Public Property ProxyPort As Integer
```

[C#]

```
public int ProxyPort {get; set;}
```

Property Value

An integer value which specifies the proxy port number.

Remarks

The **ProxyPort** property is used to set the port number that the control will use to establish a connection with the proxy server. A value of zero specifies that the client will connect to the proxy server using the standard FTP service port.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.ProxyServer Property

Gets and sets the hostname or IP address of a proxy server.

[Visual Basic]

```
Public Property ProxyServer As String
```

[C#]

```
public string ProxyServer {get; set;}
```

Property Value

A string which specifies the hostname or IP address of the proxy server that will be used when establishing a connection.

Remarks

The **ProxyServer** property should be set to the name of the proxy server that you want to connect to. This property may be set to either a fully qualified domain name, or an IP address. This property is only used if the **ProxyType** property specifies a proxy server type other than **proxyNone**.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.ProxyType Property

Gets and sets the type of proxy server the client will use to establish a connection.

[Visual Basic]

```
Public Property ProxyType As FileTransferProxy
```

[C#]

```
public FileTransfer.FileTransferProxy ProxyType {get; set;}
```

Property Value

An FileTransferProxy enumeration which specifies the type of proxy that the client will connect through.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.ProxyUser Property

Gets and sets the username used to authenticate the connection to a proxy server.

[Visual Basic]

```
Public Property ProxyUser As String
```

[C#]

```
public string ProxyUser {get; set;}
```

Property Value

A string which specifies the username.

Remarks

The **ProxyUser** property specifies the user that is logging in to the proxy server. If the proxy server does not require the user to login, then this property is ignored.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.RemoteFile Property

Gets and sets a value which specifies a file name on the server.

[Visual Basic]

```
Public Property RemoteFile As String
```

[C#]

```
public string RemoteFile {get; set;}
```

Property Value

A string which specifies a file name.

Remarks

The **RemoteFile** property is used to specify the name of a file on the server. Note that this property specifies the name of the file only, not a complete URL. To specify a complete URL, set the **URL** property and the control will automatically set the **RemoteFile** property to the correct value.

In most cases, the remote file name should be specified using an absolute path that begins with a leading slash character.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.ResultCode Property

Gets a value which specifies the last result code returned by the server.

[Visual Basic]

```
Public ReadOnly Property ResultCode As Integer
```

[C#]

```
public int ResultCode {get;}
```

Property Value

An integer value which specifies the last result code returned by the server.

Remarks

Result codes are three-digit numeric values returned by the remote server and may be broken down into the following ranges:

ResultCode	Description
100-199	Positive preliminary result. This indicates that the requested action is being initiated, and the client should expect another reply from the server before proceeding.
200-299	Positive completion result. This indicates that the server has successfully completed the requested action.
300-399	Positive intermediate result. This indicates that the requested action cannot complete until additional information is provided to the server.
400-499	Transient negative completion result. This indicates that the requested action did not take place, but the error condition is temporary and may be attempted again.
500-599	Permanent negative completion result. This indicates that the requested action did not take place.

It is important to note that while some result codes have become standardized, not all servers respond to commands using the same result codes. For example, one server may respond with a result code of 221 to indicate success, while another may respond with a value of 235. It is recommended that applications check for ranges of values to determine if a command was successful, not a specific value.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.ResultString Property

Gets a string value which describes the result of the previous command.

[Visual Basic]

```
Public ReadOnly Property ResultString As String
```

[C#]

```
public string ResultString {get;}
```

Property Value

A string which describes the result of the previous command executed on the server.

Remarks

The **ResultString** property returns the result string from the last action taken by the client. This string is generated by the remote server, and typically is used to describe the result code. For example, if an error is indicated by the result code, the result string may describe the condition that caused the error.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.Secure Property

Gets and sets a value which specifies if a secure connection is established.

[Visual Basic]

```
Public Property Secure As Boolean
```

[C#]

```
public bool Secure {get; set;}
```

Property Value

Returns **true** if a secure connection is established; otherwise returns **false**. The default value is **false**.

Remarks

The **Secure** property determines if a secure connection is established with the server. The default value for this property is **false**, which specifies that a standard connection to the server is used. To establish a secure connection, the application should set this property value to **true** prior to calling the **Connect** method. Once the connection has been established, the client may exchange data with the server as with standard connections.

It is strongly recommended that any application that sets this property **true** use error handling to trap an errors that may occur. If the class is unable to initialize the security libraries, or otherwise cannot create a secure session for the client, an exception may be generated when this property value is set to **true**.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.SecureCipher Property

Gets a value that specifies the encryption algorithm used for a secure connection.

[Visual Basic]
```
Public ReadOnly Property SecureCipher As SecureCipherAlgorithm
```

[C#]
```
public FileTransfer.SecureCipherAlgorithm SecureCipher {get;}
```

Property Value

A SecureCipherAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureCipher** property returns a value which identifies the algorithm used to encrypt the data stream. If a secure connection has not been established, this property will return a value of **cipherNone**.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.SecureHash Property

Gets a value that specifies the message digest algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureHash As SecureHashAlgorithm
```

[C#]

```
public FileTransfer.SecureHashAlgorithm SecureHash {get;}
```

Property Value

A SecureHashAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureHash** property returns a value which identifies the message digest (hash) algorithm that was selected when a secure connection was established. If a secure connection has not been established, this property will return a value of **hashNone**.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.SecureKeyExchange Property

Gets a value that specifies the key exchange algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureKeyExchange As SecureKeyAlgorithm
```

[C#]

```
public FileTransfer.SecureKeyAlgorithm SecureKeyExchange {get;}
```

Property Value

A SecureKeyAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureKeyExchange** property returns a value which identifies the key exchange algorithm that was selected when a secure connection was established. If a secure connection has not been established, this property will return a value of **keyExchangeNone**.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.SecureProtocol Property

Gets and sets a value which specifies the protocol used for a secure connection.

[Visual Basic]

```
Public Property SecureProtocol As SecurityProtocols
```

[C#]

```
public FileTransfer.SecurityProtocols SecureProtocol {get; set;}
```

Property Value

A SecurityProtocols enumeration value which identifies the protocol to be used when establishing a secure connection.

Remarks

The **SecureProtocol** property can be used to specify the security protocol to be used when establishing a secure connection with a server or accepting a secure connection from a client. By default, the class will attempt to use either SSL v3 or TLS v1 to establish the connection, with the appropriate protocol automatically selected based on the capabilities of the remote host. It is recommended that you only change this property value if you fully understand the implications of doing so. Assigning a value to this property will override the default protocol and force the class to attempt to use only the protocol specified.

Multiple security protocols may be specified by combining them using a bitwise **or** operator. After a connection has been established, this property will identify the protocol that was selected. Attempting to set this property after a connection has been established will result in an exception being thrown. This property should only be set after setting the **Secure** property to **true** and before calling the **Accept** or **Connect** methods.

In some cases, a server may only accept a secure connection if the TLS v1 protocol is specified. If the security protocol is not compatible with the server, then the connection will fail with an error indicating that the control is unable to establish a security context for the session. In this case, try assigning the property to **protocolTLS1** and attempt the connection again.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.ServerDirectory Property

Gets and sets a value which specifies the current working directory on the file server.

[Visual Basic]
```
Public Property ServerDirectory As String
```

[C#]
```
public string ServerDirectory {get; set;}
```

Property Value

A string value which specifies the current working directory on the file server. If there is no active connection to a server, or the File Transfer Protocol has not been used to establish a connection, this property will return an empty string.

Remarks

This property specifies the name of a directory on the file server. When a connection is first established to an FTP server, this property will return the current working directory. Setting this property is equivalent to using the **ChangeDirectory** method.

It is important to note that this property is only valid when connected to an FTP server. If you change the value of this property, and the current working directory cannot be changed on the server, an exception will be thrown which must be handled by the client application.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.ServerName Property

Gets and sets a value which specifies the host name used to establish a connection.

[Visual Basic]

```
Public Property ServerName As String
```

[C#]

```
public string ServerName {get; set;}
```

Property Value

A string which specifies a server domain name.

Remarks

The **ServerName** property can be used to set the host name for a remote system that you wish to communicate with.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.ServerPort Property

Gets and sets a value which specifies the remote port number.

[Visual Basic]

```
Public Property ServerPort As Integer
```

[C#]

```
public int ServerPort {get; set;}
```

Property Value

An integer value which specifies a port number.

Remarks

The **ServerPort** property is used to set the port number that will be used to establish a connection with a server.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.ServerType Property

Gets and sets a value which specifies the type of file server the client is connecting to.

[Visual Basic]
```
Public Property ServerType As FileServerType
```

[C#]
```
public FileTransfer.FileServerType ServerType {get; set;}
```

Property Value

Returns a FileServerType enumeration value which specifies the type of file server, either using the File Transfer Protocol or the Hypertext Transfer Protocol.

Remarks

If this property value is specified as **serverUnknown**, the actual server type will either be determined by the URL or the value of the **ServerPort** property, if the port number is recognized as a standard service port. If the server type cannot be automatically identified, an error will be returned when the **Connect** method is called.

If you are connecting using a non-standard port number, and are not specifying a URL for the remote file name, you should always initialize this property value to the correct server type before attempting to establish a connection.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.System Property

Gets a string value which identifies the server.

[Visual Basic]
```
Public ReadOnly Property System As String
```

[C#]
```
public string System {get;}
```

Property Value

A string which identifies the type of server the client has connected to.

Remarks

The **System** property returns information about the server operating system. This is a read-only property that can be used by the application to identify the type of server that the client has connected to. Reading this property will cause the SYST command to be sent to the server and will only return a useful value after a connection has been established with the server.

By convention, the first whitespace separated token in the string identifies the general operating system platform. For example, here are some strings commonly returned by various FTP servers:

Examples	Description
UNIX Type: L8	A standard UNIX based server. This is the most common value returned by servers, and this indicates that the server supports UNIX file naming and directory listing conventions. This string may also include additional information such as the specific variant of UNIX and its version. The L8 portion of the string is a convention that lets the client know that a byte consists of 8 bits.
Windows_NT Version 5.0	A standard Windows based server, typically part of Internet Information Services (ISS). The server will use Windows file naming and directory listing conventions. The version identifies the specific release of Windows. For example, version 4.0 specifies Windows NT 4.0 and 5.0 specifies Windows 2000.
VMS V7.1 AlphaServer	A server running the VMS operating system. The server will use the standard file naming and directory listing conventions for that platform. Note that it is possible that a VMS system may also be configured to operate in a UNIX emulation mode, in which case it will return UNIX instead of VMS.
NetWare	A server running the NetWare operating system. The server will use the standard file naming and directory listing conventions for that platform. Note that it is possible that a NetWare system may be configured to operate in a UNIX emulation

	mode, in which case it return UNIX instead of NetWare.
WORLDGROUP Type: L8	A server running the WorldGroup software on the Windows platform. This server supports UNIX file naming and directory listing conventions. WorldGroup is a collaborative workgroup, email and file sharing service which includes an FTP server.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.TaskCount Property

Get the number of active background file transfers.

[Visual Basic]

```
Public ReadOnly Property TaskCount As Integer
```

[C#]

```
public int TaskCount {get;}
```

Property Value

An integer value that specifies the number of background file transfers that are currently in progress.

Remarks

The **TaskCount** property returns the number of background file transfers that are currently in progress. One common use for this property is to create a timer that periodically checks this value when a series of background transfers are started. When the property returns a value of zero, that indicates all of the background transfers have completed. This property can also be used to enumerate the active background tasks in conjunction with the **TaskList** property.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.TaskId Property

Get the task identifier for the last background file transfer.

[Visual Basic]

```
Public ReadOnly Property TaskId As Integer
```

[C#]

```
public int TaskId {get;}
```

Property Value

An integer value the uniquely identifies the current background task.

Remarks

The **TaskId** property returns the task ID associated with the current background task. This identifies the last background file transfer that was initiated with a call to the **AsyncGetFile** or **AsyncPutFile** methods. This property value will change with each subsequent background transfer that is performed. If this property returns a value of zero, that indicates that no background tasks have been started for this instance of the class.

To enumerate the active background tasks, use the **TaskCount** property and the **TaskList** array.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.TaskList Property

Get an array of active background task identifiers.

[Visual Basic]
```
Public ReadOnly Property TaskList As ArrayList
```

[C#]
```
public System.Collections.ArrayList TaskList {get;}
```

Property Value

An **ArrayList** object that contains a list of integer values that uniquely identify the active background tasks that have been started by this instance of the class.

Remarks

The **TaskList** property returns a read-only **ArrayList** object that is popularted with the task identifiers for all active background tasks that have been created by this instance of the class. The current number of active tasks can be determined using the **TaskCount** property.

As background tasks complete and additional tasks are started, the values stored in this array will change. The application should never make any assumptions about the numeric values stored in the array or the order they are returned. Task IDs should be considered opaque values that are unique to the process. When a background task completes, its corresponding ID is removed from the list of active tasks and this can potentially change the task ID values associated with each index into the array.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.ThreadModel Property

Gets and sets a value which specifies the threading model for the class instance.

[Visual Basic]

```
Public Property ThreadModel As ThreadingModel
```

[C#]

```
public FileTransfer.ThreadingModel ThreadModel {get; set;}
```

Property Value

Returns one or more ThreadingModel enumeration value which specifies the threading model for the client. The default value for this property is **modelSingleThread**.

Remarks

The **ThreadModel** property specifies the threading model that is used by the class instance when a connection is established. The default value for this property is **modelSingleThread**, which specifies that only the thread that established the connection should be permitted to invoke methods. It is important to note that this threading model does not limit the application to a single thread of execution. When a session is established using the **Connect** method, that session is attached to the thread that created it. From that point on, until the session is terminated, only the owner may invoke methods in that instance of the class. The ownership of the class instance may be transferred from one thread to another using the **AttachThread** method.

Setting this property to **modelFreeThread** disables certain internal safety checks that are performed by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads. The application must ensure that no two threads will attempt to invoke a blocking method at the same time. In other words, if one thread invokes a method, the application must ensure that another thread will not attempt to invoke any other method at the same time using the same instance of the class.

Changing the value of this property will not affect an active client session. The threading model must be specified prior to invoking the **Connect** method.

See Also

FileTransfer Class | SocketTools Namespace | AttachThread Method | ThreadingModel Enumeration | ThreadModel Attribute

FileTransfer.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]

```
Public Property ThrowError As Boolean
```

[C#]

```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.Timeout Property

Gets and sets a value which specifies a timeout period in seconds.

[Visual Basic]

```
Public Property Timeout As Integer
```

[C#]

```
public int Timeout {get; set;}
```

Property Value

An integer value which specifies a timeout period in seconds.

Remarks

Setting the **Timeout** property specifies the number of seconds until a blocking operation fails and returns an error.

The timeout period is only used when the client is in blocking mode. Although this property can be changed when the client is in non-blocking mode, the value will be ignored until the client is returned to blocking mode.

For most applications it is recommended the timeout period be set between 10 and 20 seconds.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.TimeZone Property

Gets and sets the current timezone offset in seconds.

[Visual Basic]

```
Public Property TimeZone As Integer
```

[C#]

```
public int TimeZone {get; set;}
```

Property Value

An integer value which specifies the current timezone offset in seconds.

Remarks

The **TimeZone** property returns the current offset from UTC in seconds. Setting the property changes the current timezone offset to the specified value. The value of this property is initially determined by the date and time settings on the local system.

This property value is used in conjunction with the **Localize** property to control how date and time localization is handled.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.Trace Property

Gets and sets a value which indicates if network function logging is enabled.

[Visual Basic]

```
Public Property Trace As Boolean
```

[C#]

```
public bool Trace {get; set;}
```

Property Value

Returns **true** if network function tracing is enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Trace** property is used to enable (or disable) the tracing of network function calls. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the SocketTools networking classes and is enabled or disabled for an entire process. This means that once trace logging is enabled for a given component, all of the function calls made by the process using any of the SocketTools classes will be logged. For example, if you have an application using both the File Transfer Protocol and Post Office Protocol classes, and you set the **Trace** property to **true**, function calls made by both classes will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all classes used by the process.

If trace logging is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the logfile is fairly large. Since logfiles can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

When redistributing your application, make sure that you include the **SocketTools10.TraceLog.dll** module with your installation. If this library is not present, then no trace output will be generated and the value of the **Trace** property will be ignored. Only those function calls made by the SocketTools networking classes will be logged. Calls made directly to the Windows Sockets API, or calls made by other classes, will not be logged.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.TraceFile Property

Gets and sets a value which specifies the name of the logfile.

[Visual Basic]

```
Public Property TraceFile As String
```

[C#]

```
public string TraceFile {get; set;}
```

Property Value

A string which specifies the name of the file.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named **SocketTools.log** is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFile** property should be set to the same value for each instance. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
 MyApp INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0 MyApp WRN:
connect(46, 192.0.0.1:1234, 16) returned -1 [10035] MyApp ERR: accept(46,
NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the control is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Note that if the specified file cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.TraceFlags Property

Gets and sets a value which specifies the client function tracing flags.

[Visual Basic]

```
Public Property TraceFlags As TraceOptions
```

[C#]

```
public FileTransfer.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration which specifies the amount of detail written to the trace logfile.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Because network function tracing is enabled per-process, the trace flags are shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFlags** property should be set to the same value for each instance. Changing the trace flags for any one instance of the class will affect the logging performed for all SocketTools classes used by the application.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being written and the error **errorOperationWouldBlock** occurs, a warning is generated because the application simply needs to attempt to write the data at a later time.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.TransferBytes Property

Gets a value which specifies the number of bytes transferred to or from the remote server.

[Visual Basic]

```
Public ReadOnly Property TransferBytes As Long
```

[C#]

```
public long TransferBytes {get;}
```

Property Value

An integer value which specifies the number of bytes of data transferred to or from the server.

Remarks

The **TransferBytes** property returns the number of bytes that have been copied to or from the remote FTP server. If this property is read while a transfer is ongoing, the property returns the number of bytes that have been copied up to that point. If read after a transfer has completed, the total number of bytes copied is returned. This property value is reset with every data transfer.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.TransferTime Property

Gets a value which specifies the number of seconds elapsed during a data transfer.

[Visual Basic]

```
Public ReadOnly Property TransferTime As Integer
```

[C#]

```
public int TransferTime {get;}
```

Property Value

An integer value which specifies the transfer time in seconds.

Remarks

The **TransferTime** returns the number of seconds that have elapsed since the data connection was opened on the remote server. If the property is read while a transfer is ongoing, it returns the elapsed time. If the property is read after the transfer is complete, it returns the total number of seconds it took to transfer the data. This property value is reset with every data transfer.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.URL Property

Gets and sets the current URL used to access a file on the server.

[Visual Basic]

```
Public Property URL As String
```

[C#]

```
public string URL {get; set;}
```

Property Value

A string which specifies the current URL.

Remarks

The **URL** property returns the current Uniform Resource Locator string which is used by the control to access a file on the server. URLs have a specific format which provides information about the server, port, path and file name, as well as optional information such as a username and password for authentication:

```
ftp://[username : [password] @] remotehost [:remoteport] / [path/...]
filename [;type=id]
```

The first part of the URL is the protocol and in this case will always be "ftp", or "ftps" if a secure connection is being used. If a username and password is required for authentication, then this will be included in the URL before the name of the server; otherwise an anonymous FTP session is assumed. Next, there is the name of the server to connect to, optionally followed by a port number. If no port number is given, then the default port for the protocol will be used. This is followed by the path, and then the name of the file on the server. An optional file type may be specified as well, with the type identifier being either "a" for text files or "i" for binary files.

One important consideration when using FTP URLs is that the path is relative to the user's home directory and should not be considered an absolute path from the root directory on the server. If no username and password is provided, then an anonymous session is used and the path is relative to the public directory used by the FTP server.

Here are some typical examples of URLs used to access files on an FTP server:

```
ftp://www.example.com/pub/financial/jan2020.xls
```

In this example, the server is www.example.com, the path is "pub/financial" and the file name is "jan2020.xls". The default port will be used to access the file, and no username and password is provided for authentication so this file must be publicly available to anonymous users.

```
ftp://www.example.com:2121/employees/picnic.doc
```

In this example, the server is www.example.com, the path is "employees" and the file name is "picnic.doc". However, the client should connect to an alternative port number, in this case 2121. This file must also be available to anonymous users because no username or password has been specified.

```
ftps://executive:secret@www.example.com/corporate/projections/sales2020.xls
```

In this example, the server is www.example.com and, the path is "corporate/projections" and the file name is "sales2020.xls". Because the protocol is ftps, a secure connection on port 990 will be established. The user name "executive" and password "secret" will be used to authenticate the session.

When setting the **URL** property, the class will parse the string and automatically update the **HostName**, **RemotePort**, **UserName**, **Password**, **RemotePath** and **RemoteFile** properties according to the values specified in the URL. This enables an application to simply provide the URL and then call the **Connect** method to establish the connection.

Note that if this property is assigned a value which cannot be parsed, an exception will be thrown that indicates that the property value is invalid. If the user enters an invalid URL and there is no exception handler, the unhandled exception will terminate the application.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.UserName Property

Gets and sets the username used to authenticate the client session.

[Visual Basic]

```
Public Property UserName As String
```

[C#]

```
public string UserName {get; set;}
```

Property Value

A string which specifies the username.

Remarks

If a username is not specified when the **Connect** method is called, the value of this property will be used as the default username when establishing a connection with the server.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.Version Property

Gets a value which returns the current version of the FileTransfer class library.

[Visual Basic]

```
Public ReadOnly Property Version As String
```

[C#]

```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the FileTransfer class library. This value can be used by an application for validation and debugging purposes.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer Methods

The methods of the **FileTransfer** class are listed below. For a complete list of **FileTransfer** class members, see the FileTransfer Members topic.

Public Static (Shared) Methods

ErrorText	Returns the description of an error code.

Public Instance Methods

AddFileType	Associate a file name extension with a specific file type.
AsyncGetFile	Overloaded. Download a file from the server to the local system in the background.
AsyncPutFile	Overloaded. Upload a file from the local system to the server in the background.
AttachThread	Attach an instance of the class to the current thread
Cancel	Cancel the current blocking client operation.
ChangeDirectory	Change the current working directory on the remote server.
CloseDirectory	Close the directory that was previously opened with the OpenDirectory method.
Command	Overloaded. Send a custom command to the server.
Connect	Overloaded. Establish a connection with a file server.
DeleteFile	Delete a file on the remote server.
Disconnect	Terminate the connection with the remote server.
Dispose	Overloaded. Releases all resources used by FileTransfer.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetData	Overloaded. Transfers the contents of a file on the server and stores it in byte array.
GetDirectory	Return the current working directory.
GetFile	Overloaded. Download a file from the server to the local system.
GetFileList	Overloaded. Returns an unparsed list of files in the specified directory.
GetFilePermissions	Overloaded. Return the access permissions for a file on the remote system.
GetFileSize	Overloaded. Returns the size of the specified file

	on the remote server.
GetFileStatus	
GetFileTime	Overloaded. Returns the modification date and time for specified file on the remote server.
GetFirstFile	Overloaded. Get information about the first file in a directory listing returned by the server.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetMultipleFiles	Download multiple files from the server to the local system using a wildcard mask.
GetNextFile	Overloaded. Get information about the next file in a directory listing returned by the server.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the FileTransfer class.
Login	Overloaded. Login to the remote server.
Logout	Log the current user off the server.
MakeDirectory	Create a new directory on the server.
OpenDirectory	Overloaded. Open the specified directory on the server.
PostFile	Overloaded. Post the contents of the specified file to a script executed on the remote server.
PutData	Overloaded. Transfers data from a byte array and stores it in a file on the remote server.
PutFile	Overloaded. Upload a file from the local system to the server.
PutMultipleFiles	Upload multiple files from the local system to the server using a wildcard mask.
RemoveDirectory	Remove a directory on the server.
RenameFile	Change the name of a file on the server.
Reset	Reset the internal state of the object, resetting all properties to their default values.
SetFilePermissions	Change the access permissions for a file on the server.
SetFileTime	Changes the modification date and time for a file on the server.
TaskAbort	Overloaded. Abort the specified asynchronous task.
TaskDone	Overloaded. Determine if an asynchronous task has completed.

TaskResume	Overloaded. Resume execution of an asynchronous task.
TaskSuspend	Overloaded. Suspend execution of an asynchronous task.
TaskWait	Overloaded. Wait for an asynchronous task to complete.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
VerifyFile	Overloaded. Verify that the contents of a file on the local system are the same as the specified file on the server.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the FileTransfer class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.AddFileType Method

Associate a file name extension with a specific file type.

[Visual Basic]

```
Public Function AddFileType( _
   ByVal fileExtension As String, _
   ByVal fileType As FileTransferType _
) As Boolean
```

[C#]

```
public bool AddFileType(
   string fileExtension,
   FileTransferType fileType
);
```

Parameters

fileExtension

A string value which specifies the file name extension.

fileType

A FileTransferType enumeration which specifies the type of file associated with the file extension.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method is used to associate specific file types with file name extensions. The class has an internal list of standard text file extensions which it automatically recognizes. This method can be used to extend or modify that list for the client session.

See Also

FileTransfer Class | SocketTools Namespace | FileTransferType Enumeration

FileTransfer.AsyncGetFile Method

Download a file from the server to the local system in the background.

Overload List

Download a file from the server to the local system in the background.

public bool AsyncGetFile(string);

Download a file from the server to the local system in the background.

public bool AsyncGetFile(string,string);

Download a file from the server to the local system in the background.

public bool AsyncGetFile(string,string,long);

Download a file from the server to the local system in the background.

public bool AsyncGetFile(string,string,long,bool);

See Also

FileTransfer Class | SocketTools Namespace | AsyncPutFile Method

FileTransfer.AsyncGetFile Method (String)

Download a file from the server to the local system in the background.

```
[Visual Basic]
Overloads Public Function AsyncGetFile( _
   ByVal localFile As String _
) As Boolean
```

```
[C#]
public bool AsyncGetFile(
   string localFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be created, overwritten or appended to. The file pathing and name conventions must be that of the local host.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncGetFile** method will download the contents of a remote file to a file on the local system. It is similar to the **GetFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being downloaded from the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create a new instance of this class for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is downloaded, the class will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to download more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.AsyncGetFile Overload List | AsyncPutFile Method

FileTransfer.AsyncGetFile Method (String, String)

Download a file from the server to the local system in the background.

[Visual Basic]

```
Overloads Public Function AsyncGetFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String _
) As Boolean
```

[C#]

```
public bool AsyncGetFile(
   string localFile,
   string remoteFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be created, overwritten or appended to. The file pathing and name conventions must be that of the local host.

remoteFile

A string that specifies the file on the server that will be transferred to the local system. The file pathing and name conventions must be that of the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncGetFile** method will download the contents of a remote file to a file on the local system. It is similar to the **GetFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being downloaded from the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create a new instance of this class for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is downloaded, the class will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to download more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.AsyncGetFile Overload List | AsyncPutFile Method

---

FileTransfer.AsyncGetFile Method (String, String, Int64)

Download a file from the server to the local system in the background.

[Visual Basic]

```
Overloads Public Function AsyncGetFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal offset As Long _
) As Boolean
```

[C#]

```
public bool AsyncGetFile(
   string localFile,
   string remoteFile,
   long offset
);
```

Parameters

localFile
A string that specifies the file on the local system that will be created, overwritten or appended to. The file pathing and name conventions must be that of the local host.

remoteFile
A string that specifies the file on the server that will be transferred to the local system. The file pathing and name conventions must be that of the server.

offset
A integer value which specifies the offset where the file transfer should begin. A value of zero specifies that the file transfer should start at the beginning of the file. A value greater than zero is typically used to restart a transfer that has not completed successfully. Note that specifying a non-zero offset requires that the server support the command to restart file transfers.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncGetFile** method will download the contents of a remote file to a file on the local system. It is similar to the **GetFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being downloaded from the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create a new instance of this class for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is downloaded, the class will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to download more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.AsyncGetFile Overload List | AsyncPutFile Method

FileTransfer.AsyncGetFile Method (String, String, Int64, Boolean)

Download a file from the server to the local system in the background.

[Visual Basic]

```
Overloads Public Function AsyncGetFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal offset As Long, _
   ByVal appendFile As Boolean _
) As Boolean
```

[C#]

```
public bool AsyncGetFile(
   string localFile,
   string remoteFile,
   long offset,
   bool appendFile
);
```

Parameters

localFile
: A string that specifies the file on the local system that will be created, overwritten or appended to. The file pathing and name conventions must be that of the local host.

remoteFile
: A string that specifies the file on the server that will be transferred to the local system. The file pathing and name conventions must be that of the server.

offset
: A integer value which specifies the offset where the file transfer should begin. A value of zero specifies that the file transfer should start at the beginning of the file. A value greater than zero is typically used to restart a transfer that has not completed successfully. Note that specifying a non-zero offset requires that the server support the command to restart file transfers.

appendFile
: A boolean value which specifies if the contents of the remote file should be appended to the local file, rather than overwriting it. A value of **true** specifies that the local file should be appended to. A value of **false** specifies that the local file should be overwritten. This parameter is only used with the File Transfer Protocol.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncGetFile** method will download the contents of a remote file to a file on the local system. It is similar to the **GetFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being downloaded from the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create a new instance of this class for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is downloaded, the class will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to download more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.AsyncGetFile Overload List | AsyncPutFile Method

FileTransfer.AsyncPutFile Method

Upload a file from the local system to the server in the background.

Overload List

Upload a file from the local system to the server in the background.

public bool AsyncPutFile(string);

Upload a file from the local system to the server in the background.

public bool AsyncPutFile(string,string);

Upload a file from the local system to the server in the background.

public bool AsyncPutFile(string,string,long);

Upload a file from the local system to the server in the background.

public bool AsyncPutFile(string,string,long,bool);

See Also

FileTransfer Class | SocketTools Namespace | AsyncGetFile Method

FileTransfer.AsyncPutFile Method (String)

Upload a file from the local system to the server in the background.

[Visual Basic]

```
Overloads Public Function AsyncPutFile( _
   ByVal localFile As String _
) As Boolean
```

[C#]

```
public bool AsyncPutFile(
   string localFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncPutFile** method will upload the contents of a file on the local system to the server. It is similar to the **PutFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being uploaded to the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create an instance of the control for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is uploaded, the control will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to upload more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.AsyncPutFile Overload List | AsyncGetFile Method

FileTransfer.AsyncPutFile Method (String, String)

Upload a file from the local system to the server in the background.

[Visual Basic]

```
Overloads Public Function AsyncPutFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String _
) As Boolean
```

[C#]

```
public bool AsyncPutFile(
   string localFile,
   string remoteFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

remoteFile

A string that specifies the file on the server that will be created, overwritten or appended to. The file pathing and name conventions must be that of the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncPutFile** method will upload the contents of a file on the local system to the server. It is similar to the **PutFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being uploaded to the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create an instance of the control for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is uploaded, the control will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to upload more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.AsyncPutFile Overload List | AsyncGetFile Method

FileTransfer.AsyncPutFile Method (String, String, Int64)

Upload a file from the local system to the server in the background.

[Visual Basic]

```
Overloads Public Function AsyncPutFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal offset As Long _
) As Boolean
```

[C#]

```
public bool AsyncPutFile(
   string localFile,
   string remoteFile,
   long offset
);
```

Parameters

localFile
A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

remoteFile
A string that specifies the file on the server that will be created, overwritten or appended to. The file pathing and name conventions must be that of the server.

offset
A integer value which specifies the offset where the file transfer should begin. A value of zero specifies that the file transfer should start at the beginning of the file. A value greater than zero is typically used to restart a transfer that has not completed successfully. Note that specifying a non-zero offset requires that the server support the command to restart file transfers.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncPutFile** method will upload the contents of a file on the local system to the server. It is similar to the **PutFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being uploaded to the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create an instance of the control for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is uploaded, the control will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to upload more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.AsyncPutFile Overload List | AsyncGetFile Method

FileTransfer.AsyncPutFile Method (String, String, Int64, Boolean)

Upload a file from the local system to the server in the background.

[Visual Basic]

```
Overloads Public Function AsyncPutFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal offset As Long, _
   ByVal appendFile As Boolean _
) As Boolean
```

[C#]

```
public bool AsyncPutFile(
   string localFile,
   string remoteFile,
   long offset,
   bool appendFile
);
```

Parameters

localFile
A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

remoteFile
A string that specifies the file on the server that will be created, overwritten or appended to. The file pathing and name conventions must be that of the server.

offset
A integer value which specifies the offset where the file transfer should begin. A value of zero specifies that the file transfer should start at the beginning of the file. A value greater than zero is typically used to restart a transfer that has not completed successfully. Note that specifying a non-zero offset requires that the server support the command to restart file transfers.

appendFile
A boolean value which specifies if the contents of the remote file should be appended to the local file, rather than overwriting it. A value of **true** specifies that the local file should be appended to. A value of **false** specifies that the local file should be overwritten. This parameter is only used with the File Transfer Protocol.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncPutFile** method will upload the contents of a file on the local system to the server. It is similar to the **PutFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being uploaded to the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create an instance of the control for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is uploaded, the control will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to upload more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.AsyncPutFile Overload List | AsyncGetFile Method

FileTransfer.AttachThread Method

Attach an instance of the class to the current thread

[Visual Basic]
```
Public Function AttachThread() As Boolean
```

[C#]
```
public bool AttachThread();
```

Return Value

A boolean value which specifies if the client could be attached to the current thread. If this method returns **false**, the client could not be attached to the thread and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

When an instance of the class is created it is associated with the current thread that created it. Normally, if another thread attempts to perform an operation using that instance, an error is returned. This is used to ensure that other threads cannot interfere with an operation being performed by the owner thread. In some cases, it may be desirable for one thread in an application to create an instance of the class, establish a connection and then pass that instance to another worker thread. The **AttachThread** method can be used to change the ownership of the class instance to the new worker thread.

This method should be called by the new thread immediately after it has been created, and if the new thread does not release the handle itself, the ownership of the handle should be restored by the original thread. Under no circumstances should **AttachThread** be used to forcibly destroy an instance of a class allocated by another thread while a blocking operation is in progress. To cancel a blocking operation, use the **Cancel** method and then delete the class instance after the blocking function exits and control is returned to the current thread.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.Cancel Method

Cancel the current blocking client operation.

[Visual Basic]

```
Public Sub Cancel()
```

[C#]

```
public void Cancel();
```

Remarks

When the **Cancel** method is called, the blocking client operation will not immediately fail. An internal flag is set which causes the blocking operation to exit with an error. This means that the application cannot cancel an operation and immediately perform some other blocking function. Instead it must allow the calling stack to unwind, returning back to the blocking operation before making any further function calls.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.ChangeDirectory Method

Change the current working directory on the remote server.

[Visual Basic]
```
Public Function ChangeDirectory( _
   ByVal pathName As String _
) As Boolean
```

[C#]
```
public bool ChangeDirectory(
   string pathName
);
```

Parameters

pathName
A string which specifies the directory on the remote server. The file pathing and name conventions must be that of the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.CloseDirectory Method

Close the directory that was previously opened with the OpenDirectory method.

[Visual Basic]

```
Public Function CloseDirectory() As Boolean
```

[C#]

```
public bool CloseDirectory();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.Command Method

Send a custom command to the server.

Overload List

Send a custom command to the server.

public bool Command(string);

Send a custom command to the server.

public bool Command(string,string);

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.Command Method (String)

Send a custom command to the server.

[Visual Basic]

```
Overloads Public Function Command( _
   ByVal command As String _
) As Boolean
```

[C#]

```
public bool Command(
   string command
);
```

Parameters

command

A string which specifies the command to send. Valid commands vary based on the Internet protocol and the type of server that the client is connected to. Consult the protocol standard and/or the technical reference documentation for the server to determine what commands may be issued by a client application.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Command** method sends a command to the server and processes the result code sent back in response to that command. This method can be used to send custom commands to a server to take advantage of features or capabilities that may not be supported internally by the class library.

To determine the specific status code returned by the server, check the value of the **ResultCode** property after the method returns.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.Command Overload List

FileTransfer.Command Method (String, String)

Send a custom command to the server.

[Visual Basic]

```
Overloads Public Function Command( _
   ByVal command As String, _
   ByVal parameters As String _
) As Boolean
```

[C#]

```
public bool Command(
   string command,
   string parameters
);
```

Parameters

command

A string which specifies the command to send. Valid commands vary based on the Internet protocol and the type of server that the client is connected to. Consult the protocol standard and/or the technical reference documentation for the server to determine what commands may be issued by a client application.

parameters

An string which specifies one or more parameters to be sent along with the command. If more than one parameter is required, they must be separated by a single space character. Consult the protocol standard and/or technical reference documentation for the server to determine what parameters should be provided when issuing a specific command.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Command** method sends a command to the server and processes the result code sent back in response to that command. This method can be used to send custom commands to a server to take advantage of features or capabilities that may not be supported internally by the class library.

To determine the specific status code returned by the server, check the value of the **ResultCode** property after the method returns.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.Command Overload List

FileTransfer.Connect Method

Establish a connection with a file server.

Overload List

Establish a connection with a file server.

public bool Connect();

Establish a connection with a file server.

public bool Connect(string);

Establish a connection with a file server.

public bool Connect(string,int);

Establish a connection with a file server.

public bool Connect(string,int,string,string);

Establish a connection with a file server.

public bool Connect(string,int,string,string,int);

Establish a connection with a file server.

public bool Connect(string,int,string,string,int,FileTransferOptions);

Establish a connection with a file server.

public bool Connect(string,int,string,string,string);

Establish a connection with a file server.

public bool Connect(string,int,string,string,string,int);

Establish a connection with a file server.

public bool Connect(string,int,string,string,string,int,FileTransferOptions);

Establish a connection with a file server.

public bool Connect(string,string,string);

See Also

FileTransfer Class | SocketTools Namespace | ServerName Property | ServerPort Property | ServerType Property | URL Property

FileTransfer.Connect Method ()

Establish a connection with a file server.

[Visual Basic]

```
Overloads Public Function Connect() As Boolean
```

[C#]

```
public bool Connect();
```

Return Value

A boolean value which specifies if the connection has been established. A return value of **true** indicates that the connection has completed and the application may transfer files between the local system and the server. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Connect** method is used to establish a connection with the server. It is the first method that must be called prior to the application transferring files or issuing commands to the server. If a connection already exists, the current connection will be closed and a new connection will be established.

It is permissible to specify a complete URL as the first argument to the method and the connection will be established with the server using specified protocol. Passing a complete URL to the **Connect** method has the same effect as setting the **URL** property and then calling the method with no arguments.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.Connect Overload List | ServerName Property | ServerPort Property | ServerType Property | URL Property

FileTransfer.Connect Method (String)

Establish a connection with a file server.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName
);
```

Parameters

hostName

A string which specifies the server to establish a connection with. This may specify a host name or an Internet address in dot-notation.

Return Value

A boolean value which specifies if the connection has been established. A return value of **true** indicates that the connection has completed and the application may transfer files between the local system and the server. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Connect** method is used to establish a connection with the server. It is the first method that must be called prior to the application transferring files or issuing commands to the server. If a connection already exists, the current connection will be closed and a new connection will be established.

It is permissible to specify a complete URL as the first argument to the method and the connection will be established with the server using specified protocol. Passing a complete URL to the **Connect** method has the same effect as setting the **URL** property and then calling the method with no arguments.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.Connect Overload List | ServerName Property | ServerPort Property | ServerType Property | URL Property

FileTransfer.Connect Method (String, Int32)

Establish a connection with a file server.

```
[Visual Basic]
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer _
) As Boolean
```

```
[C#]
public bool Connect(
   string hostName,
   int hostPort
);
```

Parameters

hostName

A string which specifies the server to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

Return Value

A boolean value which specifies if the connection has been established. A return value of **true** indicates that the connection has completed and the application may transfer files between the local system and the server. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Connect** method is used to establish a connection with the server. It is the first method that must be called prior to the application transferring files or issuing commands to the server. If a connection already exists, the current connection will be closed and a new connection will be established.

It is permissible to specify a complete URL as the first argument to the method and the connection will be established with the server using specified protocol. Passing a complete URL to the **Connect** method has the same effect as setting the **URL** property and then calling the method with no arguments.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.Connect Overload List | ServerName Property | ServerPort Property | ServerType Property | URL Property

FileTransfer.Connect Method (String, Int32, String, String)

Establish a connection with a file server.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword
);
```

Parameters

hostName
A string which specifies the server to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort
An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName
A string which specifies the user name which will be used to authenticate the client session. If the user name is specified as an empty string, then the login is considered to be anonymous.

userPassword
A string which specifies the password which will be used to authenticate the client session. This argument may be an empty string if no password is required for the specified user, or if no username has been specified.

Return Value

A boolean value which specifies if the connection has been established. A return value of **true** indicates that the connection has completed and the application may transfer files between the local system and the server. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Connect** method is used to establish a connection with the server. It is the first method that must be called prior to the application transferring files or issuing commands to the server. If a connection already exists, the current connection will be closed and a new connection will be established.

It is permissible to specify a complete URL as the first argument to the method and the connection will be established with the server using specified protocol. Passing a complete URL to the **Connect** method has the same effect as setting the **URL** property and then calling the method with no arguments.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.Connect Overload List | ServerName Property | ServerPort Property | ServerType Property | URL Property

FileTransfer.Connect Method (String, Int32, String, String, Int32)

Establish a connection with a file server.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   int timeout
);
```

Parameters

hostName

A string which specifies the server to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName

A string which specifies the user name which will be used to authenticate the client session. If the user name is specified as an empty string, then the login is considered to be anonymous.

userPassword

A string which specifies the password which will be used to authenticate the client session. This argument may be an empty string if no password is required for the specified user, or if no username has been specified.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller.

Return Value

A boolean value which specifies if the connection has been established. A return value of **true** indicates that the connection has completed and the application may transfer files between the local system and the server. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Connect** method is used to establish a connection with the server. It is the first method that must be called prior to the application transferring files or issuing commands to the server. If a connection already exists, the current connection will be closed and a new connection will be established.

It is permissible to specify a complete URL as the first argument to the method and the connection will be established with the server using specified protocol. Passing a complete URL to the **Connect** method has the same effect as setting the **URL** property and then calling the method with no arguments.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.Connect Overload List | ServerName Property | ServerPort Property | ServerType Property | URL Property

FileTransfer.Connect Method (String, Int32, String, String, Int32, FileTransferOptions)

Establish a connection with a file server.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal timeout As Integer, _
   ByVal options As FileTransferOptions _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   int timeout,
   FileTransferOptions options
);
```

Parameters

hostName

A string which specifies the server to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName

A string which specifies the user name which will be used to authenticate the client session. If the user name is specified as an empty string, then the login is considered to be anonymous.

userPassword

A string which specifies the password which will be used to authenticate the client session. This argument may be an empty string if no password is required for the specified user, or if no username has been specified.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller.

options

One or more of the FileTransferOptions enumeration flags.

Return Value

A boolean value which specifies if the connection has been established. A return value of **true** indicates that the connection has completed and the application may transfer files between the local system and the server. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Connect** method is used to establish a connection with the server. It is the first method that must be called prior to the application transferring files or issuing commands to the server. If a connection already exists, the current connection will be closed and a new connection will be established.

It is permissible to specify a complete URL as the first argument to the method and the connection will be established with the server using specified protocol. Passing a complete URL to the **Connect** method has the same effect as setting the **URL** property and then calling the method with no arguments.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.Connect Overload List | ServerName Property | ServerPort Property | ServerType Property | URL Property

FileTransfer.Connect Method (String, Int32, String, String, String)

Establish a connection with a file server.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal userAccount As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   string userAccount
);
```

Parameters

hostName

A string which specifies the server to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName

A string which specifies the user name which will be used to authenticate the client session. If the user name is specified as an empty string, then the login is considered to be anonymous.

userPassword

A string which specifies the password which will be used to authenticate the client session. This argument may be an empty string if no password is required for the specified user, or if no username has been specified.

userAccount

A string that specifies the account name to be used to authenticate the current client session. This parameter may be an empty string if no account name is required for the specified user.

Return Value

A boolean value which specifies if the connection has been established. A return value of **true** indicates that the connection has completed and the application may transfer files between the local system and the server. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Connect** method is used to establish a connection with the server. It is the first method that must be called prior to the application transferring files or issuing commands to the server. If a connection already exists, the current connection will be closed and a new connection will be established.

It is permissible to specify a complete URL as the first argument to the method and the connection will be established with the server using specified protocol. Passing a complete URL to the **Connect** method has the same effect as setting the **URL** property and then calling the method with no arguments.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.Connect Overload List | ServerName Property | ServerPort Property | ServerType Property | URL Property

FileTransfer.Connect Method (String, Int32, String, String, String, Int32)

Establish a connection with a file server.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal userAccount As String, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   string userAccount,
   int timeout
);
```

Parameters

hostName
A string which specifies the server to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort
An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName
A string which specifies the user name which will be used to authenticate the client session. If the user name is specified as an empty string, then the login is considered to be anonymous.

userPassword
A string which specifies the password which will be used to authenticate the client session. This argument may be an empty string if no password is required for the specified user, or if no username has been specified.

userAccount
A string that specifies the account name to be used to authenticate the current client session. This parameter may be an empty string if no account name is required for the specified user. Note that this parameter is only used with the File Transfer Protocol.

timeout
An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller.

Return Value

A boolean value which specifies if the connection has been established. A return value of **true** indicates that the connection has completed and the application may transfer files between the local system and the server. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Connect** method is used to establish a connection with the server. It is the first method that must be called prior to the application transferring files or issuing commands to the server. If a connection already exists, the current connection will be closed and a new connection will be established.

It is permissible to specify a complete URL as the first argument to the method and the connection will be established with the server using specified protocol. Passing a complete URL to the **Connect** method has the same effect as setting the **URL** property and then calling the method with no arguments.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.Connect Overload List | ServerName Property | ServerPort Property | ServerType Property | URL Property

FileTransfer.Connect Method (String, Int32, String, String, String, Int32, FileTransferOptions)

Establish a connection with a file server.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal userAccount As String, _
   ByVal timeout As Integer, _
   ByVal options As FileTransferOptions _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   string userAccount,
   int timeout,
   FileTransferOptions options
);
```

Parameters

hostName

A string which specifies the server to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName

A string which specifies the user name which will be used to authenticate the client session. If the user name is specified as an empty string, then the login is considered to be anonymous.

userPassword

A string which specifies the password which will be used to authenticate the client session. This argument may be an empty string if no password is required for the specified user, or if no username has been specified.

userAccount

A string that specifies the account name to be used to authenticate the current client session. This parameter may be an empty string if no account name is required for the specified user. Note that this parameter is only used with the File Transfer Protocol.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller.

options

One or more of the FileTransferOptions enumeration flags.

Return Value

A boolean value which specifies if the connection has been established. A return value of **true** indicates that the connection has completed and the application may transfer files between the local system and the server. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Connect** method is used to establish a connection with the server. It is the first method that must be called prior to the application transferring files or issuing commands to the server. If a connection already exists, the current connection will be closed and a new connection will be established.

It is permissible to specify a complete URL as the first argument to the method and the connection will be established with the server using specified protocol. Passing a complete URL to the **Connect** method has the same effect as setting the **URL** property and then calling the method with no arguments.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.Connect Overload List | ServerName Property | ServerPort Property | ServerType Property | URL Property

FileTransfer.Connect Method (String, String, String)

Establish a connection with a file server.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   string userName,
   string userPassword
);
```

Parameters

hostName

A string which specifies the server to establish a connection with. This may specify a host name or an Internet address in dot-notation.

userName

A string which specifies the user name which will be used to authenticate the client session. If the user name is specified as an empty string, then the login is considered to be anonymous.

userPassword

A string which specifies the password which will be used to authenticate the client session. This argument may be an empty string if no password is required for the specified user, or if no username has been specified.

Return Value

A boolean value which specifies if the connection has been established. A return value of **true** indicates that the connection has completed and the application may transfer files between the local system and the server. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Connect** method is used to establish a connection with the server. It is the first method that must be called prior to the application transferring files or issuing commands to the server. If a connection already exists, the current connection will be closed and a new connection will be established.

It is permissible to specify a complete URL as the first argument to the method and the connection will be established with the server using specified protocol. Passing a complete URL to the **Connect** method has the same effect as setting the **URL** property and then calling the method with no arguments.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.Connect Overload List | ServerName Property | ServerPort Property | ServerType Property | URL Property

FileTransfer.DeleteFile Method

Delete a file on the remote server.

[Visual Basic]

```
Public Function DeleteFile( _
   ByVal remoteFile As String _
) As Boolean
```

[C#]

```
public bool DeleteFile(
   string remoteFile
);
```

Parameters

remoteFile

A string which specifies the name of the file on the remote server that is to be deleted. The file pathing and name conventions must be that of the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **DeleteFile** method deletes an existing file from the remote server. The user must have the appropriate permission to delete the specified file.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.Disconnect Method

Terminate the connection with the remote server.

[Visual Basic]
```
Public Sub Disconnect()
```

[C#]
```
public void Disconnect();
```

Remarks

The **Disconnect** method terminates the connection with the remote host and closes the handle allocated by the class. Note that the handle is not immediately released when the connection is terminated and will enter a wait state for two minutes. After the time wait period has elapsed, the handle will be released by the operating system. This is a normal safety mechanism to handle any packets that may arrive after the connection has been closed.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.Dispose Method

Releases all resources used by FileTransfer.

Overload List

Releases all resources used by FileTransfer.

public void Dispose();

Releases the unmanaged resources allocated by the FileTransfer class and optionally releases the managed resources.

protected virtual void Dispose(bool);

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.Dispose Method ()

Releases all resources used by FileTransfer.

[Visual Basic]

```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]

```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.Dispose Overload List

FileTransfer.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the FileTransfer class and optionally releases the managed resources.

[Visual Basic]

```
Overridable Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]

```
protected virtual void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **FileTransfer** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.Dispose Overload List

FileTransfer.Finalize Method

Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.

[Visual Basic]

```
Overrides Protected Sub Finalize()
```

[C#]

```
protected override void Finalize();
```

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.GetDirectory Method

Return the current working directory.

[Visual Basic]

```
Public Function GetDirectory( _
   ByRef pathName As String _
) As Boolean
```

[C#]

```
public bool GetDirectory(
   ref string pathName
);
```

Parameters

pathName

A string passed by reference which will contain the current working directory on the server when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.GetFile Method

Download a file from the server to the local system.

Overload List

Download a file from the server to the local system.

public bool GetFile();

Download a file from the server to the local system.

public bool GetFile(string);

Download a file from the server to the local system.

public bool GetFile(string,string);

Download a file from the server to the local system.

public bool GetFile(string,string,bool);

Download a file from the server to the local system.

public bool GetFile(string,string,long);

Download a file from the server to the local system.

public bool GetFile(string,string,long,bool);

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.GetFile Method ()

Download a file from the server to the local system.

[Visual Basic]

```
Overloads Public Function GetFile() As Boolean
```

[C#]

```
public bool GetFile();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFile** method copies an existing file from the server to the local system. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.GetFile Overload List

FileTransfer.GetFile Method (String)

Download a file from the server to the local system.

[Visual Basic]

```
Overloads Public Function GetFile( _
   ByVal localFile As String _
) As Boolean
```

[C#]

```
public bool GetFile(
   string localFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be created, overwritten or appended to. The file pathing and name conventions must be that of the local host.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFile** method copies an existing file from the server to the local system. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.GetFile Overload List

FileTransfer.GetFile Method (String, String)

Download a file from the server to the local system.

[Visual Basic]

```
Overloads Public Function GetFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String _
) As Boolean
```

[C#]

```
public bool GetFile(
   string localFile,
   string remoteFile
);
```

Parameters

localFile
A string that specifies the file on the local system that will be created, overwritten or appended to. The file pathing and name conventions must be that of the local host.

remoteFile
A string that specifies the file on the remote system that will be transferred to the local system. The file pathing and name conventions must be that of the remote host.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFile** method copies an existing file from the server to the local system. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.GetFile Overload List

FileTransfer.GetFile Method (String, String, Boolean)

Download a file from the server to the local system.

[Visual Basic]

```
Overloads Public Function GetFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal appendFile As Boolean _
) As Boolean
```

[C#]

```
public bool GetFile(
   string localFile,
   string remoteFile,
   bool appendFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be created, overwritten or appended to. The file pathing and name conventions must be that of the local host.

remoteFile

A string that specifies the file on the remote system that will be transferred to the local system. The file pathing and name conventions must be that of the remote host.

appendFile

A boolean value which specifies if the contents of the remote file should be appended to the local file, rather than overwriting it. A value of **true** specifies that the local file should be appended to. A value of **false** specifies that the local file should be overwritten. This parameter is only used with the File Transfer Protocol.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFile** method copies an existing file from the server to the local system. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.GetFile Overload List

FileTransfer.GetFile Method (String, String, Int64)

Download a file from the server to the local system.

[Visual Basic]

```
Overloads Public Function GetFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal offset As Long _
) As Boolean
```

[C#]

```
public bool GetFile(
   string localFile,
   string remoteFile,
   long offset
);
```

Parameters

localFile

A string that specifies the file on the local system that will be created, overwritten or appended to. The file pathing and name conventions must be that of the local host.

remoteFile

A string that specifies the file on the remote system that will be transferred to the local system. The file pathing and name conventions must be that of the remote host.

offset

A integer value which specifies the offset where the file transfer should begin. A value of zero specifies that the file transfer should start at the beginning of the file. A value greater than zero is typically used to restart a transfer that has not completed successfully. Note that specifying a non-zero offset requires that the server support the command to restart file transfers.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFile** method copies an existing file from the server to the local system. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.GetFile Overload List

FileTransfer.GetFile Method (String, String, Int64, Boolean)

Download a file from the server to the local system.

[Visual Basic]

```
Overloads Public Function GetFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal offset As Long, _
   ByVal appendFile As Boolean _
) As Boolean
```

[C#]

```
public bool GetFile(
   string localFile,
   string remoteFile,
   long offset,
   bool appendFile
);
```

Parameters

localFile
A string that specifies the file on the local system that will be created, overwritten or appended to. The file pathing and name conventions must be that of the local host.

remoteFile
A string that specifies the file on the remote system that will be transferred to the local system. The file pathing and name conventions must be that of the remote host.

offset
A integer value which specifies the offset where the file transfer should begin. A value of zero specifies that the file transfer should start at the beginning of the file. A value greater than zero is typically used to restart a transfer that has not completed successfully. Note that specifying a non-zero offset requires that the server support the command to restart file transfers.

appendFile
A boolean value which specifies if the contents of the remote file should be appended to the local file, rather than overwriting it. A value of **true** specifies that the local file should be appended to. A value of **false** specifies that the local file should be overwritten. This parameter is only used with the File Transfer Protocol.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFile** method copies an existing file from the server to the local system. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.GetFile Overload List

FileTransfer.GetFilePermissions Method

Return the access permissions for a file on the remote system.

Overload List

Return the access permissions for a file on the remote system.

public bool GetFilePermissions(ref FilePermissions);

Return the access permissions for a file on the remote system.

public bool GetFilePermissions(string,ref FilePermissions);

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.GetFilePermissions Method (FilePermissions)

Return the access permissions for a file on the remote system.

[Visual Basic]

```
Overloads Public Function GetFilePermissions( _
   ByRef filePerms As FilePermissions _
) As Boolean
```

[C#]

```
public bool GetFilePermissions(
   ref FilePermissions filePerms
);
```

Parameters

filePerms

An FilePermissions enumeration value which is passed by reference and set to the file permissions when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFilePermissions** method returns information about the access permissions for a specific file on the server. This method uses the STAT command to retrieve information about the specified file. If the server does not support the use of this command, an error will be returned. You can use the **Features** property to determine what features are available and/or enabled on the server.

Note that on some systems, the STAT command will not return information on files that contain spaces or tabs in the filename. In this case, the method will fail.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.GetFilePermissions Overload List

FileTransfer.GetFilePermissions Method (String, FilePermissions)

Return the access permissions for a file on the remote system.

[Visual Basic]

```
Overloads Public Function GetFilePermissions( _
   ByVal remoteFile As String, _
   ByRef filePerms As FilePermissions _
) As Boolean
```

[C#]

```
public bool GetFilePermissions(
   string remoteFile,
   ref FilePermissions filePerms
);
```

Parameters

remoteFile

A string that specifies the name of the file that the access permissions are to be returned for. The filename cannot contain any wildcard characters.

filePerms

An FilePermissions enumeration value which is passed by reference and set to the file permissions when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFilePermissions** method returns information about the access permissions for a specific file on the server. This method uses the STAT command to retrieve information about the specified file. If the server does not support the use of this command, an error will be returned. You can use the **Features** property to determine what features are available and/or enabled on the server.

Note that on some systems, the STAT command will not return information on files that contain spaces or tabs in the filename. In this case, the method will fail.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.GetFilePermissions Overload List

FileTransfer.GetFileSize Method

Returns the size of the specified file on the remote server.

Overload List

Returns the size of the specified file on the remote server.

public bool GetFileSize(ref int);

Returns the size of the specified file on the remote server.

public bool GetFileSize(ref long);

Returns the size of the specified file on the remote server.

public bool GetFileSize(string,ref int);

Returns the size of the specified file on the remote server.

public bool GetFileSize(string,ref long);

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.GetFileSize Method (String, Int32)

Returns the size of the specified file on the remote server.

[Visual Basic]

```
Overloads Public Function GetFileSize( _
   ByVal remoteFile As String, _
   ByRef fileSize As Integer _
) As Boolean
```

[C#]

```
public bool GetFileSize(
   string remoteFile,
   ref int fileSize
);
```

Parameters

remoteFile

A string that specifies the name of the file on the server. The filename cannot contain any wildcard characters and must follow the naming conventions of the operating system the server is hosted on.

fileSize

An integer value which is passed by reference and will specify the size of the file on the server when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method returns the size of the specified file in bytes. If an FTP connection has been established, the server must support the SIZE command. If this command is not supported, the method will fail. You can use the **Features** property to determine what features are available and/or enabled on the server.

Note that if the file on the server is a text file, it is possible that the value returned by this method will not match the size of the file when it is downloaded to the local system. This is because different operating systems use different sequences of characters to mark the end of a line of text, and when a file is transferred in text mode, the end of line character sequence is automatically converted to a carriage return-linefeed, which is the convention used by the Windows platform.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.GetFileSize Overload List

FileTransfer.GetFileSize Method (String, Int64)

Returns the size of the specified file on the remote server.

[Visual Basic]

```
Overloads Public Function GetFileSize( _
   ByVal remoteFile As String, _
   ByRef fileSize As Long _
) As Boolean
```

[C#]

```
public bool GetFileSize(
   string remoteFile,
   ref long fileSize
);
```

Parameters

remoteFile
: A string that specifies the name of the file on the server. The filename cannot contain any wildcard characters and must follow the naming conventions of the operating system the server is hosted on.

fileSize
: An integer value which is passed by reference and will specify the size of the file on the server when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method returns the size of the specified file in bytes. If an FTP connection has been established, the server must support the SIZE command. If this command is not supported, the method will fail. You can use the **Features** property to determine what features are available and/or enabled on the server.

Note that if the file on the server is a text file, it is possible that the value returned by this method will not match the size of the file when it is downloaded to the local system. This is because different operating systems use different sequences of characters to mark the end of a line of text, and when a file is transferred in text mode, the end of line character sequence is automatically converted to a carriage return-linefeed, which is the convention used by the Windows platform.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.GetFileSize Overload List

FileTransfer.GetFileStatus Method

[Visual Basic]

```
Public Function GetFileStatus( _
   ByVal remoteFile As String, _
   ByRef fileInfo As FileInformation _
) As Boolean
```

[C#]

```
public bool GetFileStatus(
   string remoteFile,
   ref FileInformation fileInfo
);
```

Parameters

remoteFile

A string which specifies the name of the file that status information is to be returned for.

fileInfo

A FileInformation structure that is passed by reference. When the method returns, the members of this structure will be populated with information about the file on the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFileStatus** method returns information about the specified file. The filename must be specified using the server file naming conventions, and cannot include wildcard characters. The primary difference between using this method and using the **OpenDirectory, GetFirstFile** and **GetNextFile** methods to obtain file information is that the file status information is returned on the command channel. This method cannot be used while a file transfer is in progress or while a file listing is being returned by the server.

This method requires that the server return file status information in response to the STAT command. Some servers, for example on VMS platforms, do not provide this information. On some systems, the STAT command will not return information on files that contain spaces or tabs in the filename. In this case, the method will return an empty structure.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.GetFileTime Method

Returns the modification date and time for specified file on the remote server.

Overload List

Returns the modification date and time for specified file on the remote server.

public bool GetFileTime(string,ref DateTime);

Returns the modification date and time for specified file on the remote server.

public bool GetFileTime(string,ref DateTime,bool);

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.GetFileTime Method (String, DateTime)

Returns the modification date and time for specified file on the remote server.

[Visual Basic]

```
Overloads Public Function GetFileTime( _
   ByVal remoteFile As String, _
   ByRef fileDate As Date _
) As Boolean
```

[C#]

```
public bool GetFileTime(
   string remoteFile,
   ref DateTime fileDate
);
```

Parameters

remoteFile

A string that specifies the name of the file on the server. The filename cannot contain any wildcard characters and must follow the naming conventions of the operating system the server is hosted on.

fileDate

A **System.DateTime** structure which is passed by reference. When the method returns, this object will be set to the date and time that the file was created or last modified.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method uses the MTDM command to determine the modification time for the file. If the server does not support this command, the method will attempt to use the STAT command to determine the file modification time. You can use the **Features** property to determine what features are available and/or enabled on the server.

The value of the Localize property determines if the date and time are adjusted for the local timezone.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.GetFileTime Overload List

FileTransfer.GetFileTime Method (String, DateTime, Boolean)

Returns the modification date and time for specified file on the remote server.

[Visual Basic]

```
Overloads Public Function GetFileTime( _
   ByVal remoteFile As String, _
   ByRef fileDate As Date, _
   ByVal localDate As Boolean _
) As Boolean
```

[C#]

```
public bool GetFileTime(
   string remoteFile,
   ref DateTime fileDate,
   bool localDate
);
```

Parameters

remoteFile

A string that specifies the name of the file on the server. The filename cannot contain any wildcard characters and must follow the naming conventions of the operating system the server is hosted on.

fileDate

A **System.DateTime** structure which is passed by reference. When the method returns, this object will be set to the date and time that the file was created or last modified.

localDate

A boolean flag which specifies if the date and time for the file should adjusted for the local timezone. A value of **true** specifies that the date and time should be adjusted for the local timezone. A value of **false** specifies that the date and time should be returned as a UTC (Coordinated Universal Time) value.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method uses the MTDM command to determine the modification time for the file. If the server does not support this command, the method will attempt to use the STAT command to determine the file modification time. You can use the **Features** property to determine what features are available and/or enabled on the server.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.GetFileTime Overload List

FileTransfer.GetFirstFile Method

Get information about the first file in a directory listing returned by the server.

Overload List

Get information about the first file in a directory listing returned by the server.

public bool GetFirstFile(ref FileInformation);

Get the first file name in a directory listing returned by the server.

public bool GetFirstFile(ref string);

Get information about the first file in a directory listing returned by the server.

public bool GetFirstFile(ref string,ref bool);

Get information about the first file in a directory listing returned by the server.

public bool GetFirstFile(ref string,ref long,ref bool);

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.GetFirstFile Method (FileInformation)

Get information about the first file in a directory listing returned by the server.

[Visual Basic]

```
Overloads Public Function GetFirstFile( _
   ByRef fileInfo As FileInformation _
) As Boolean
```

[C#]

```
public bool GetFirstFile(
   ref FileInformation fileInfo
);
```

Parameters

fileInfo

A FileInformation structure that is passed by reference. When the method returns, the members of this structure will be populated with information about the file on the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

A file listing is initiated by calling the **OpenDirectory** method. Then, the application must call **GetFirstFile**, followed by calling **GetNextFile** in a loop until the method returns **false**. Once the complete directory listing has been returned, the **CloseDirectory** method must be called.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.GetFirstFile Overload List

FileTransfer.GetFirstFile Method (String)

Get the first file name in a directory listing returned by the server.

[Visual Basic]

```
Overloads Public Function GetFirstFile( _
   ByRef fileName As String _
) As Boolean
```

[C#]

```
public bool GetFirstFile(
   ref string fileName
);
```

Parameters

fileName
A string passed by reference which will contain a file name when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

A file listing is initiated by calling the **OpenDirectory** method. Then, the application must call **GetFirstFile**, followed by calling **GetNextFile** in a loop until the method returns **false**. Once the complete directory listing has been returned, the **CloseDirectory** method must be called.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.GetFirstFile Overload List

FileTransfer.GetFirstFile Method (String, Boolean)

Get information about the first file in a directory listing returned by the server.

[Visual Basic]

```
Overloads Public Function GetFirstFile( _
   ByRef fileName As String, _
   ByRef isDirectory As Boolean _
) As Boolean
```

[C#]

```
public bool GetFirstFile(
   ref string fileName,
   ref bool isDirectory
);
```

Parameters

fileName

A string passed by reference which will contain a file name when the method returns.

isDirectory

A boolean passed by reference which will specify if the file is a regular file or a directory. A value of **true** indicates that the file is a directory. A value of **false** indicates that it is a regular file.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

A file listing is initiated by calling the **OpenDirectory** method. Then, the application must call **GetFirstFile**, followed by calling **GetNextFile** in a loop until the method returns **false**. Once the complete directory listing has been returned, the **CloseDirectory** method must be called.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.GetFirstFile Overload List

FileTransfer.GetFirstFile Method (String, Int64, Boolean)

Get information about the first file in a directory listing returned by the server.

[Visual Basic]

```
Overloads Public Function GetFirstFile( _
   ByRef fileName As String, _
   ByRef fileSize As Long, _
   ByRef isDirectory As Boolean _
) As Boolean
```

[C#]

```
public bool GetFirstFile(
   ref string fileName,
   ref long fileSize,
   ref bool isDirectory
);
```

Parameters

fileName

A string passed by reference which will contain a file name when the method returns.

fileSize

An integer passed by reference which will contain the size of the file when the method returns.

isDirectory

A boolean passed by reference which will specify if the file is a regular file or a directory. A value of **true** indicates that the file is a directory. A value of **false** indicates that it is a regular file.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

A file listing is initiated by calling the **OpenDirectory** method. Then, the application must call **GetFirstFile**, followed by calling **GetNextFile** in a loop until the method returns **false**. Once the complete directory listing has been returned, the **CloseDirectory** method must be called.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.GetFirstFile Overload List

FileTransfer.GetMultipleFiles Method

Download multiple files from the server to the local system using a wildcard mask.

[Visual Basic]

```
Public Function GetMultipleFiles( _
   ByVal localPath As String, _
   ByVal remotePath As String, _
   ByVal fileMask As String _
) As Boolean
```

[C#]

```
public bool GetMultipleFiles(
   string localPath,
   string remotePath,
   string fileMask
);
```

Parameters

localPath
A string argument which specifies the name of the directory on the local system where the files will be stored. If a file by the same name already exists, it will be overwritten

remotePath
A string argument which specifies the name of the directory on the remote system where the files will be copied from. You must have permission to read the contents of the directory.

fileMask
An string argument which specifies the wildcard mask to be used when selecting what files should be transferred. Typically, this argument is a wildcard mask that limits the files downloaded from the server to those which match a specific extension.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMultipleFiles** method copies multiple files from the remote system to the local system. If the local file already exists, it is overwritten. This method will cause the current thread to block until all of the files have been transferred, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.GetNextFile Method

Get information about the next file in a directory listing returned by the server.

Overload List

Get information about the next file in a directory listing returned by the server.

public bool GetNextFile(ref FileInformation);

Get the next file name in a directory listing returned by the server.

public bool GetNextFile(ref string);

Get information about the next file in a directory listing returned by the server.

public bool GetNextFile(ref string,ref bool);

Get information about the next file in a directory listing returned by the server.

public bool GetNextFile(ref string,ref long,ref bool);

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.GetNextFile Method (FileInformation)

Get information about the next file in a directory listing returned by the server.

[Visual Basic]

```
Overloads Public Function GetNextFile( _
   ByRef fileInfo As FileInformation _
) As Boolean
```

[C#]

```
public bool GetNextFile(
   ref FileInformation fileInfo
);
```

Parameters

fileInfo

A FileInformation structure that is passed by reference. When the method returns, the members of this structure will be populated with information about the file on the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

A file listing is initiated by calling the **OpenDirectory** method. Then, the application must call **GetFirstFile**, followed by calling **GetNextFile** in a loop until the method returns **false**. Once the complete directory listing has been returned, the **CloseDirectory** method must be called.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.GetNextFile Overload List

FileTransfer.GetNextFile Method (String)

Get the next file name in a directory listing returned by the server.

[Visual Basic]

```
Overloads Public Function GetNextFile( _
   ByRef fileName As String _
) As Boolean
```

[C#]

```
public bool GetNextFile(
   ref string fileName
);
```

Parameters

fileName
A string passed by reference which will contain a file name when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

A file listing is initiated by calling the **OpenDirectory** method. Then, the application must call **GetFirstFile**, followed by calling **GetNextFile** in a loop until the method returns **false**. Once the complete directory listing has been returned, the **CloseDirectory** method must be called.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.GetNextFile Overload List

FileTransfer.GetNextFile Method (String, Boolean)

Get information about the next file in a directory listing returned by the server.

[Visual Basic]

```
Overloads Public Function GetNextFile( _
   ByRef fileName As String, _
   ByRef isDirectory As Boolean _
) As Boolean
```

[C#]

```
public bool GetNextFile(
   ref string fileName,
   ref bool isDirectory
);
```

Parameters

fileName

A string passed by reference which will contain a file name when the method returns.

isDirectory

A boolean passed by reference which will specify if the file is a regular file or a directory. A value of **true** indicates that the file is a directory. A value of **false** indicates that it is a regular file.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

A file listing is initiated by calling the **OpenDirectory** method. Then, the application must call **GetFirstFile**, followed by calling **GetNextFile** in a loop until the method returns **false**. Once the complete directory listing has been returned, the **CloseDirectory** method must be called.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.GetNextFile Overload List

FileTransfer.GetNextFile Method (String, Int64, Boolean)

Get information about the next file in a directory listing returned by the server.

[Visual Basic]

```
Overloads Public Function GetNextFile( _
   ByRef fileName As String, _
   ByRef fileSize As Long, _
   ByRef isDirectory As Boolean _
) As Boolean
```

[C#]

```
public bool GetNextFile(
   ref string fileName,
   ref long fileSize,
   ref bool isDirectory
);
```

Parameters

fileName

A string passed by reference which will contain a file name when the method returns.

fileSize

An integer passed by reference which will contain the size of the file when the method returns.

isDirectory

A boolean passed by reference which will specify if the file is a regular file or a directory. A value of **true** indicates that the file is a directory. A value of **false** indicates that it is a regular file.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

A file listing is initiated by calling the **OpenDirectory** method. Then, the application must call **GetFirstFile**, followed by calling **GetNextFile** in a loop until the method returns **false**. Once the complete directory listing has been returned, the **CloseDirectory** method must be called.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.GetNextFile Overload List

FileTransfer.Initialize Method

Initialize an instance of the FileTransfer class.

Overload List

Initialize an instance of the FileTransfer class.

public bool Initialize();

Initialize an instance of the FileTransfer class.

public bool Initialize(string);

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.Initialize Method ()

Initialize an instance of the FileTransfer class.

[Visual Basic]
```
Overloads Public Function Initialize() As Boolean
```

[C#]
```
public bool Initialize();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Initialize** method can be used to explicitly initialize an instance of the **FileTransfer** class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the **FileTransfer** class

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.Initialize Overload List

FileTransfer.Initialize Method (String)

Initialize an instance of the FileTransfer class.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Initialize** method can be used to explicitly initialize an instance of the **FileTransfer** class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the **FileTransfer** class

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.Initialize Overload List

FileTransfer.Login Method

Login to the remote server.

Overload List

Login to the remote server.

public bool Login();

Login to the remote server.

public bool Login(string,string);

Login to the remote server.

public bool Login(string,string,string);

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.Login Method ()

Login to the remote server.

[Visual Basic]
```
Overloads Public Function Login() As Boolean
```

[C#]
```
public bool Login();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Login** method identifies the user to the remote server. The value of the **UserName** and **Password** properties will be used to authenticate the client session. If the user name or password is invalid, an error will occur. By default, when a connection is established, the user is automatically authenticated. This method is typically used if you wish to log in as another user during the same session.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.Login Overload List

FileTransfer.Login Method (String, String)

Login to the remote server.

```
[Visual Basic]
Overloads Public Function Login( _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

```
[C#]
public bool Login(
   string userName,
   string userPassword
);
```

Parameters

userName
A string that specifies the name of the user logging into the server.

userPassword
A string that specifies the password used to authenticate the user.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Login** method identifies the user to the remote server. If the user name or password is invalid, an error will occur. By default, when a connection is established, the **UserName** and **Password** properties are used to automatically log the user in to the server. This method is typically used if you wish to log in as another user during the same session.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.Login Overload List

FileTransfer.Login Method (String, String, String)

Login to the remote server.

[Visual Basic]

```
Overloads Public Function Login( _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal userAccount As String _
) As Boolean
```

[C#]

```
public bool Login(
   string userName,
   string userPassword,
   string userAccount
);
```

Parameters

userName

A string that specifies the name of the user logging into the server.

userPassword

A string that specifies the password used to authenticate the user.

userAccount

A string that specifies the account name to be used when authenticating the user.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Login** method identifies the user to the remote server. If the user name or password is invalid, an error will occur. By default, when a connection is established, the **UserName**, **Password** and **Account** properties are used to automatically log the user in to the server. This method is typically used if you wish to log in as another user during the same session.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.Login Overload List

FileTransfer.Logout Method

Log the current user off the server.

[Visual Basic]
```
Public Function Logout() As Boolean
```

[C#]
```
public bool Logout();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Logout** method logs the current user off the server. The **Login** method may then be used to login as another user during the same session. Note that this method will not terminate the connection with the server.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.MakeDirectory Method

Create a new directory on the server.

[Visual Basic]

```
Public Function MakeDirectory( _
   ByVal pathName As String _
) As Boolean
```

[C#]

```
public bool MakeDirectory(
   string pathName
);
```

Parameters

pathName

A string that specifies the name of the directory to create on the server. The naming and pathing conventions used for the directory must be compatible with what is used on the operating system that hosts the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

Servers may not support creating multiple subdirectories in a single call, so applications should not assume that this can be done. For example, an error may be returned by the server if the new directory name "/Projects/Today" is specified, but the "/Projects" directory does not already exist.

It is also important to note that files and directories on UNIX based systems are case sensitive, so the directory names "Projects" and "projects" refer to two different directories. This is not the case on Windows systems, where either name would refer to the same directory.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.OpenDirectory Method

Open the current working directory on the server.

Overload List

Open the current working directory on the server.

public bool OpenDirectory();

Open the specified directory on the server.

public bool OpenDirectory(string);

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.OpenDirectory Method ()

Open the current working directory on the server.

[Visual Basic]

```
Overloads Public Function OpenDirectory() As Boolean
```

[C#]

```
public bool OpenDirectory();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **OpenDirectory** method opens the current working directory on the server so that the list of files in that directory may be obtained using the **GetFirstFile** and **GetNextFile** methods. Once all of the files in the directory have been read, the application must call the **CloseDirectory** method in order to close the data channel to the server. Failure to do this will result in an error the next time the application attempts to transfer a file or open another directory.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.OpenDirectory Overload List

FileTransfer.OpenDirectory Method (String)

Open the specified directory on the server.

[Visual Basic]

```
Overloads Public Function OpenDirectory( _
   ByVal pathName As String _
) As Boolean
```

[C#]

```
public bool OpenDirectory(
   string pathName
);
```

Parameters

pathName

A string that specifies the name of the directory to open on the server. The naming and pathing conventions used for the directory must be compatible with what is used on the operating system that hosts the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **OpenDirectory** method opens the specified directory on the server so that the list of files in that directory may be obtained using the **GetFirstFile** and **GetNextFile** methods. Once all of the files in the directory have been read, the application must call the **CloseDirectory** method in order to close the data channel to the server. Failure to do this will result in an error the next time the application attempts to transfer a file or open another directory.

Note that files and directories on UNIX based systems are case sensitive, so the directory names "Projects" and "projects" refer to two different directories. This is not the case on Windows systems, where either name would refer to the same directory.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.OpenDirectory Overload List

FileTransfer.PostFile Method

Post the contents of the specified file to a script executed on the remote server.

Overload List

Post the contents of the specified file to a script executed on the remote server.

public bool PostFile();

Post the contents of the specified file to a script executed on the remote server.

public bool PostFile(string);

Post the contents of the specified file to a script executed on the remote server.

public bool PostFile(string,string);

Post the contents of the specified file to a script executed on the remote server.

public bool PostFile(string,string,string);

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.PostFile Method ()

Post the contents of the specified file to a script executed on the remote server.

[Visual Basic]

```
Overloads Public Function PostFile() As Boolean
```

[C#]

```
public bool PostFile();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PostFile** method posts the contents of a file to a script that is executed on the server, and can only be used when connected to a web server. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The **PostFile** method is similar to the **PutFile** method in that it can be used to upload the contents of a local file to a server. However, instead of using the HTTP PUT command, the POST command is used to send the file data to a script that is executed on the server. This method has the advantage of not requiring any special configuration settings on the server, however it does require that the script be able to process multipart/form-data as defined in RFC 2388.

To support uploading files from a form on a webpage, the FILE input type is used along with the action that specifies the script that will accept the file data and process it. For example, the HTML code could look like this:

```
 <form action="/cgi-bin/upload.cgi" method="post" enctype="multipart/form-
data"> <input type="file" name="datafile" size="20"> <input type="submit">
</form>
```

In this example, the script /cgi-bin/upload.cgi is responsible for processing the file data that is posted by the client, and the form field name "datafile" is used. The user can select a file, and when the Submit button is clicked, the file data is posted to the script. To simulate this using the **PostFile** method, the ***fileName*** argument should be set to the name of the local file that is to be posted to the server. The ***resourceName*** argument should be the name of the script, in this case "/cgi-bin/upload.cgi". The ***fieldName*** argument should be specified as the string "datafile" to match the name of the field used by the form.

Note that the **PostFile** function always submits the file contents as multipart/form-data with the content type set to application/octet-stream. The script that accepts the posted data must be able to parse the multipart header block and correctly process 8-bit data. If the script assumes that the data will be posted using a specific encoding type such as base64, then the file data may not be accepted or may be corrupted by the script.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.PostFile Overload List

FileTransfer.PostFile Method (String)

Post the contents of the specified file to a script executed on the remote server.

[Visual Basic]

```
Overloads Public Function PostFile( _
   ByVal localFile As String _
) As Boolean
```

[C#]

```
public bool PostFile(
   string localFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PostFile** method posts the contents of a file to a script that is executed on the server, and can only be used when connected to a web server. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The **PostFile** method is similar to the **PutFile** method in that it can be used to upload the contents of a local file to a server. However, instead of using the HTTP PUT command, the POST command is used to send the file data to a script that is executed on the server. This method has the advantage of not requiring any special configuration settings on the server, however it does require that the script be able to process multipart/form-data as defined in RFC 2388.

To support uploading files from a form on a webpage, the FILE input type is used along with the action that specifies the script that will accept the file data and process it. For example, the HTML code could look like this:

```
 <form action="/cgi-bin/upload.cgi" method="post" enctype="multipart/form-
data"> <input type="file" name="datafile" size="20"> <input type="submit">
</form>
```

In this example, the script /cgi-bin/upload.cgi is responsible for processing the file data that is posted by the client, and the form field name "datafile" is used. The user can select a file, and when the Submit button is clicked, the file data is posted to the script. To simulate this using the **PostFile** method, the ***fileName*** argument should be set to the name of the local file that is to be posted to the server. The ***resourceName*** argument should be the name of the script, in this case "/cgi-bin/upload.cgi". The ***fieldName*** argument should be specified as the string "datafile" to match the name of the field used by the form.

Note that the **PostFile** function always submits the file contents as multipart/form-data with the content type set to application/octet-stream. The script that accepts the posted data must be able to parse the multipart header block and correctly process 8-bit data. If the script assumes that the data will be posted using a specific encoding type such as base64, then the file data may not be accepted or may be corrupted by the script.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.PostFile Overload List

FileTransfer.PostFile Method (String, String)

Post the contents of the specified file to a script executed on the remote server.

[Visual Basic]

```
Overloads Public Function PostFile( _
   ByVal localFile As String, _
   ByVal resourceName As String _
) As Boolean
```

[C#]

```
public bool PostFile(
   string localFile,
   string resourceName
);
```

Parameters

localFile
: A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

resourceName
: A string that specifies the resource on the server that the data will be posted to. Typically this is the name of an executable script on the server. The resource name should be specified using an absolute path that begins with a leading slash character.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PostFile** method posts the contents of a file to a script that is executed on the server, and can only be used when connected to a web server. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The **PostFile** method is similar to the **PutFile** method in that it can be used to upload the contents of a local file to a server. However, instead of using the HTTP PUT command, the POST command is used to send the file data to a script that is executed on the server. This method has the advantage of not requiring any special configuration settings on the server, however it does require that the script be able to process multipart/form-data as defined in RFC 2388.

To support uploading files from a form on a webpage, the FILE input type is used along with the action that specifies the script that will accept the file data and process it. For example, the HTML code could look like this:

```
 <form action="/cgi-bin/upload.cgi" method="post" enctype="multipart/form-
data"> <input type="file" name="datafile" size="20"> <input type="submit">
</form>
```

In this example, the script /cgi-bin/upload.cgi is responsible for processing the file data that is posted by the client, and the form field name "datafile" is used. The user can select a file, and when the Submit button is clicked, the file data is posted to the script. To simulate this using the **PostFile** method, the *fileName* argument should be set to the name of the local file that is to be posted to the server. The *resourceName* argument should be the name of the script, in this case "/cgi-bin/upload.cgi". The *fieldName* argument should be specified as the string "datafile" to match the name of the field used by the form.

Note that the **PostFile** function always submits the file contents as multipart/form-data with the content type set to application/octet-stream. The script that accepts the posted data must be able to parse the multipart header block and correctly process 8-bit data. If the script assumes that the data will be posted using a specific encoding type such as base64, then the file data may not be accepted or may be corrupted by the script.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.PostFile Overload List

FileTransfer.PostFile Method (String, String, String)

Post the contents of the specified file to a script executed on the remote server.

[Visual Basic]

```
Overloads Public Function PostFile( _
   ByVal localFile As String, _
   ByVal resourceName As String, _
   ByVal fieldName As String _
) As Boolean
```

[C#]

```
public bool PostFile(
   string localFile,
   string resourceName,
   string fieldName
);
```

Parameters

localFile

A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

resourceName

A string that specifies the resource on the server that the data will be posted to. Typically this is the name of an executable script on the server. The resource name should be specified using an absolute path that begins with a leading slash character.

fieldName

A string argument that specifies the form field name that the script expects. If this argument is omitted or is an empty string, a default field name of "File1" is used.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PostFile** method posts the contents of a file to a script that is executed on the server, and can only be used when connected to a web server. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The **PostFile** method is similar to the **PutFile** method in that it can be used to upload the contents of a local file to a server. However, instead of using the HTTP PUT command, the POST command is used to send the file data to a script that is executed on the server. This method has the advantage of not requiring any special configuration settings on the server, however it does require that the script be able to process multipart/form-data as defined in RFC 2388.

To support uploading files from a form on a webpage, the FILE input type is used along with the action that specifies the script that will accept the file data and process it. For example, the HTML code could look like this:

```
 <form action="/cgi-bin/upload.cgi" method="post" enctype="multipart/form-
data"> <input type="file" name="datafile" size="20"> <input type="submit">
```

```
        </form>
```

In this example, the script /cgi-bin/upload.cgi is responsible for processing the file data that is posted by the client, and the form field name "datafile" is used. The user can select a file, and when the Submit button is clicked, the file data is posted to the script. To simulate this using the **PostFile** method, the ***fileName*** argument should be set to the name of the local file that is to be posted to the server. The ***resourceName*** argument should be the name of the script, in this case "/cgi-bin/upload.cgi". The ***fieldName*** argument should be specified as the string "datafile" to match the name of the field used by the form.

Note that the **PostFile** function always submits the file contents as multipart/form-data with the content type set to application/octet-stream. The script that accepts the posted data must be able to parse the multipart header block and correctly process 8-bit data. If the script assumes that the data will be posted using a specific encoding type such as base64, then the file data may not be accepted or may be corrupted by the script.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.PostFile Overload List

FileTransfer.PutFile Method

Upload a file from the local system to the server.

Overload List

Upload a file from the local system to the server.

public bool PutFile();

Upload a file from the local system to the server.

public bool PutFile(string);

Upload a file from the local system to the server.

public bool PutFile(string,string);

Upload a file from the local system to the server.

public bool PutFile(string,string,bool);

Upload a file from the local system to the server.

public bool PutFile(string,string,long);

Upload a file from the local system to the server.

public bool PutFile(string,string,long,bool);

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.PutFile Method ()

Upload a file from the local system to the server.

[Visual Basic]
```
Overloads Public Function PutFile() As Boolean
```

[C#]
```
public bool PutFile();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutFile** method copies an existing file from the local system to the server. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.PutFile Overload List

FileTransfer.PutFile Method (String)

Upload a file from the local system to the server.

[Visual Basic]

```
Overloads Public Function PutFile( _
   ByVal localFile As String _
) As Boolean
```

[C#]

```
public bool PutFile(
   string localFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutFile** method copies an existing file from the local system to the server. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.PutFile Overload List

FileTransfer.PutFile Method (String, String)

Upload a file from the local system to the server.

[Visual Basic]

```
Overloads Public Function PutFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String _
) As Boolean
```

[C#]

```
public bool PutFile(
   string localFile,
   string remoteFile
);
```

Parameters

localFile
A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

remoteFile
A string that specifies the file on the server that will be created, overwritten or appended to. The file pathing and name conventions must be that of the remote host.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutFile** method copies an existing file from the local system to the server. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.PutFile Overload List

FileTransfer.PutFile Method (String, String, Boolean)

Upload a file from the local system to the server.

[Visual Basic]
```
Overloads Public Function PutFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal appendFile As Boolean _
) As Boolean
```

[C#]
```
public bool PutFile(
   string localFile,
   string remoteFile,
   bool appendFile
);
```

Parameters

localFile
A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

remoteFile
A string that specifies the file on the server that will be created, overwritten or appended to. The file pathing and name conventions must be that of the remote host.

appendFile
A boolean value which specifies if the contents of the local file should be appended to the remote file, rather than overwriting it. A value of **true** specifies that the remote file should be appended to. A value of **false** specifies that the remote file should be overwritten. This parameter is only used with the File Transfer Protocol.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutFile** method copies an existing file from the local system to the server. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.PutFile Overload List

FileTransfer.PutFile Method (String, String, Int64)

Upload a file from the local system to the server.

[Visual Basic]

```
Overloads Public Function PutFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal offset As Long _
) As Boolean
```

[C#]

```
public bool PutFile(
   string localFile,
   string remoteFile,
   long offset
);
```

Parameters

localFile
: A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

remoteFile
: A string that specifies the file on the server that will be created, overwritten or appended to. The file pathing and name conventions must be that of the remote host.

offset
: A numeric value which specifies the byte offset where the file transfer should begin. The default value of zero specifies that the file transfer should start at the beginning of the file. A value greater than zero is typically used to restart a transfer that has not completed successfully. Note that specifying a non-zero offset requires that the server support the REST command to restart transfers.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutFile** method copies an existing file from the local system to the server. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.PutFile Overload List

FileTransfer.PutFile Method (String, String, Int64, Boolean)

Upload a file from the local system to the server.

[Visual Basic]

```
Overloads Public Function PutFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal offset As Long, _
   ByVal appendFile As Boolean _
) As Boolean
```

[C#]

```
public bool PutFile(
   string localFile,
   string remoteFile,
   long offset,
   bool appendFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

remoteFile

A string that specifies the file on the server that will be created, overwritten or appended to. The file pathing and name conventions must be that of the remote host.

offset

A numeric value which specifies the byte offset where the file transfer should begin. The default value of zero specifies that the file transfer should start at the beginning of the file. A value greater than zero is typically used to restart a transfer that has not completed successfully. Note that specifying a non-zero offset requires that the server support the REST command to restart transfers.

appendFile

A boolean value which specifies if the contents of the local file should be appended to the remote file, rather than overwriting it. A value of **true** specifies that the remote file should be appended to. A value of **false** specifies that the remote file should be overwritten. This parameter is only used with the File Transfer Protocol.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutFile** method copies an existing file from the local system to the server. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.PutFile Overload List

FileTransfer.PutMultipleFiles Method

Upload multiple files from the local system to the server using a wildcard mask.

[Visual Basic]

```
Public Function PutMultipleFiles( _
   ByVal localPath As String, _
   ByVal remotePath As String, _
   ByVal fileMask As String _
) As Boolean
```

[C#]

```
public bool PutMultipleFiles(
   string localPath,
   string remotePath,
   string fileMask
);
```

Parameters

localPath
A string argument which specifies the name of the directory on the local system where the files will be copied from. You must have permission to read the contents of the directory.

remotePath
A string argument which specifies the name of the directory on the remote system where the files will be stored. You must have permission to modify the contents of the directory and create files.

fileMask
A string argument which specifies the wildcard mask to be used when selecting what files should be transferred. An empty string indicates that all files in the specified directory should be uploaded. Typically, this argument is a wildcard mask that limits the files uploaded to the server to those which match a specific extension.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutMultipleFiles** method copies multiple files from the local system to the remote server. If the remote file already exists, it is overwritten. This method will cause the current thread to block until all of the files have been transferred, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.RemoveDirectory Method

Remove a directory on the server.

```
[Visual Basic]
Public Function RemoveDirectory( _
   ByVal pathName As String _
) As Boolean
```

```
[C#]
public bool RemoveDirectory(
   string pathName
);
```

Parameters

pathName

A string that specifies the name of the directory to remove from the server. The naming and pathing conventions used for the directory must be compatible with what is used on the operating system that hosts the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **RemoveDirectory** method removes an existing directory on the remote host. You must have the appropriate permission to remove the directory, or an error will occur. Note that most operating systems will not permit you to remove a directory that contains files or other subdirectories.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.RenameFile Method

Change the name of a file on the server.

[Visual Basic]

```
Public Function RenameFile( _
   ByVal oldName As String, _
   ByVal newName As String _
) As Boolean
```

[C#]

```
public bool RenameFile(
   string oldName,
   string newName
);
```

Parameters

oldName

A string that specifies the name of the file to be renamed on the server. The file must exist on the server, otherwise an error will be returned.

newName

A string that specifies the new name for the file on the server. The naming conventions used for the file must be compatible with what is used on the operating system that hosts the server. Note that some servers may not permit you to rename the file if a file with the new name already exists.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **RenameFile** method changes the name of an existing file on the server to a new name. Note that you must have permission to rename the file or an error will occur. On UNIX based systems this means that you must have write permission to the directory where the file is being renamed.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. If a connection to a server has been established, it will be terminated the resources allocated for the client session will be released. All properties will be reset to their default values.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.SetFilePermissions Method

Change the access permissions for a file on the server.

[Visual Basic]

```
Public Function SetFilePermissions( _
   ByVal remoteFile As String, _
   ByVal filePerms As FilePermissions _
) As Boolean
```

[C#]

```
public bool SetFilePermissions(
   string remoteFile,
   FilePermissions filePerms
);
```

Parameters

remoteFile
A string that specifies the name of the file that the access permissions are to be returned for. The filename cannot contain any wildcard characters.

filePerms
An FilePermissions enumeration which specifies the new permissions for the file.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method uses the SITE CHMOD command to set the permissions for the file. This command is typically only supported on servers that are hosted on UNIX based systems. If the command is not supported, an error will be returned. You can use the **Features** property to determine what features are available and/or enabled on the server.

Users who are familiar with the UNIX operating system will recognize the **chmod** command used to change the file permissions. However, it should be noted that the numeric value used as an argument to the command is in octal, not decimal. For example, issuing the command **chmod 644 filename.txt** on a UNIX based system will make the file readable and writable by the owner, and readable by other users in the owner's group as well as all other users. The value 644 is an octal value, which is equivalent to the decimal value 420. If you were to mistakenly specify 644 as the value for the *Permissions* argument, rather than the decimal value of 420, the permissions on the file would be incorrect. It is strongly recommended that you use the enumeration values and do not cast a numeric value as the argument.

Note that Visual Basic allows you to specify an integer value in octal by prefixing it with &O. For example, &O644 could be used as the file permissions value. C# and C++ consider any integer with a preceding 0 to be an octal number, so 0644 would be a valid permissions value.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.SetFileTime Method

Changes the modification date and time for a file on the server.

[Visual Basic]

```
Public Function SetFileTime( _
   ByVal remoteFile As String, _
   ByVal fileDate As Date _
) As Boolean
```

[C#]

```
public bool SetFileTime(
   string remoteFile,
   DateTime fileDate
);
```

Parameters

remoteFile

A string that specifies the name of the file on the server. The filename cannot contain any wildcard characters and must follow the naming conventions of the operating system the server is hosted on.

fileDate

A **System.DateTime** value that specifies the new date and time for the file.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SetFileTime** method changes the modification date and time for the specified file on the remote server. This method uses the MTDM command to change the modification time for the file. If the server does not support this command, the method will return an error. Note that some servers only support the MDTM command to return, but not change, the file modification time.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.TaskAbort Method

Abort all asynchronous tasks that are currently active.

Overload List

Abort all asynchronous tasks that are currently active.

public bool TaskAbort();

Abort the specified asynchronous task.

public bool TaskAbort(int);

Abort the specified asynchronous task.

public bool TaskAbort(int,int);

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.TaskAbort Method ()

Abort all asynchronous tasks that are currently active.

[Visual Basic]

```
Overloads Public Function TaskAbort() As Boolean
```

[C#]

```
public bool TaskAbort();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskAbort** method signals all background worker threads created by this instance of the class to abort their current operation and terminate as soon as possible. This version of the method will signal each active task and return immediately to the caller.

The **Reset** and **Uninitialize** methods will abort all active background transfers and wait for those tasks to complete before returning to the caller. It is recommended that your application explicitly wait for background transfers to complete or abort them using this method before allowing the program to terminate. This will ensure that your program can perform any necessary cleanup operations. If there are active background tasks running at the time that the class instance is disposed, it can force the instance to stop those worker threads immediately without waiting for them to terminate gracefully.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.TaskAbort Overload List

FileTransfer.TaskAbort Method (Int32)

Abort the specified asynchronous task.

[Visual Basic]

```
Overloads Public Function TaskAbort( _
   ByVal taskId As Integer _
) As Boolean
```

[C#]

```
public bool TaskAbort(
   int taskId
);
```

Parameters

taskId

An integer value that specifies the unique identifier associated with a background task.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskAbort** method signals the background worker thread associated with the task ID to abort the current operation and terminate as soon as possible. This version of the method returns immediately after the background thread has been signaled.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.TaskAbort Overload List

FileTransfer.TaskAbort Method (Int32, Int32)

Abort the specified asynchronous task.

[Visual Basic]

```
Overloads Public Function TaskAbort( _
   ByVal taskId As Integer, _
   ByVal timeWait As Integer _
) As Boolean
```

[C#]

```
public bool TaskAbort(
   int taskId,
   int timeWait
);
```

Parameters

taskId

An integer value that specifies the unique identifier associated with a background task.

timeWait

An integer value that specifies the number of milliseconds to wait for the background task to abort.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskAbort** method signals the background worker thread associated with the task ID to abort the current operation and terminate as soon as possible. If the ***timeWait*** parameter has a value of zero, the method returns immediately after the background thread has been signaled. If the ***timeWait*** parameter is non-zero, the method will wait that amount of time for the background thread to terminate.

The **Reset** and **Uninitialize** methods will abort all active background transfers and wait for those tasks to complete before returning to the caller. It is recommended that your application explicitly wait for background transfers to complete or abort them using this method before allowing the program to terminate. This will ensure that your program can perform any necessary cleanup operations. If there are active background tasks running at the time that the class instance is disposed, it can force the instance to stop those worker threads immediately without waiting for them to terminate gracefully.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.TaskAbort Overload List

FileTransfer.TaskDone Method

Determine if the current asynchronous task has completed.

Overload List

Determine if the current asynchronous task has completed.

public bool TaskDone();

Determine if an asynchronous task has completed.

public bool TaskDone(int);

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.TaskDone Method ()

Determine if the current asynchronous task has completed.

[Visual Basic]

```
Overloads Public Function TaskDone() As Boolean
```

[C#]

```
public bool TaskDone();
```

Return Value

This method returns a Boolean value. If the task has finished, the return value is **true**. If the background task is still active, the return value is **false**.

Remarks

The **TaskDone** method is used to determine if the current asynchronous task has completed. This overloaded version of the method is functionally equivalent to providing the value of the **TaskId** property as the unique task identifier.

If you use this method to poll the status of a background task from within the main UI thread, you must ensure that Windows messages are processed so that the application remains responsive to the end-user. To check if a background transfer has completed, it is recommended that you use a timer to periodically call this method rather than calling it repeatedly within a loop.

To determine if the task completed successfully, the **TaskWait** method will provide the last error code associated with the task. Note that if this method returns **true**, it is guaranteed that calling **TaskWait** using the same task ID will return the error code to the caller immediately without causing the current thread to block.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.TaskDone Overload List

FileTransfer.TaskDone Method (Int32)

Determine if an asynchronous task has completed.

[Visual Basic]

```
Overloads Public Function TaskDone( _
   ByVal taskId As Integer _
) As Boolean
```

[C#]

```
public bool TaskDone(
   int taskId
);
```

Parameters

taskId

An optional integer value that specifies the unique identifier associated with a background task.

Return Value

This method returns a Boolean value. If the task has finished, the return value is **true**. If the background task is still active, the return value is **false**.

Remarks

The **TaskDone** method is used to determine if the specified asynchronous task has completed.

If you use this method to poll the status of a background task from within the main UI thread, you must ensure that Windows messages are processed so that the application remains responsive to the end-user. To check if a background transfer has completed, it is recommended that you use a timer to periodically call this method rather than calling it repeatedly within a loop.

To determine if the task completed successfully, the **TaskWait** method will provide the last error code associated with the task. Note that if this method returns **true**, it is guaranteed that calling **TaskWait** using the same task ID will return the error code to the caller immediately without causing the current thread to block.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.TaskDone Overload List

FileTransfer.TaskResume Method

Resume execution of the current asynchronous task.

Overload List

Resume execution of the current asynchronous task.

public bool TaskResume();

Resume execution of an asynchronous task.

public bool TaskResume(int);

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.TaskResume Method ()

Resume execution of the current asynchronous task.

[Visual Basic]

```
Overloads Public Function TaskResume() As Boolean
```

[C#]

```
public bool TaskResume();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskResume** method resumes execution of the current background task that was previously suspended using the **TaskSuspend** method. This overloaded version of the method is functionally equivalent to providing the value of the **TaskId** property as the unique task identifier.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.TaskResume Overload List

FileTransfer.TaskResume Method (Int32)

Resume execution of an asynchronous task.

[Visual Basic]

```
Overloads Public Function TaskResume( _
   ByVal taskId As Integer _
) As Boolean
```

[C#]

```
public bool TaskResume(
   int taskId
);
```

Parameters

taskId

An integer value that specifies the unique identifier associated with a background task.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskResume** method resumes execution of the background worker thread that was previously suspended using the **TaskSuspend** method.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.TaskResume Overload List

FileTransfer.TaskSuspend Method

Suspend execution of the current asynchronous task.

Overload List

Suspend execution of the current asynchronous task.

public bool TaskSuspend();

Suspend execution of an asynchronous task.

public bool TaskSuspend(int);

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.TaskSuspend Method ()

Suspend execution of the current asynchronous task.

[Visual Basic]

```
Overloads Public Function TaskSuspend() As Boolean
```

[C#]

```
public bool TaskSuspend();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskSuspend** method will suspend execution of the background worker thread associated with the current task. This overloaded version of the method is functionally equivalent to providing the value of the **TaskId** property as the unique task identifier.

Once the task has been suspended, it will no longer be scheduled for execution, however the client session will remain active and the task may be resumed using the **TaskResume** method. Note that if a task is suspended for a long period of time, the background operation may fail because it has exceeded the timeout period imposed by the server.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.TaskSuspend Overload List

FileTransfer.TaskSuspend Method (Int32)

Suspend execution of an asynchronous task.

[Visual Basic]

```
Overloads Public Function TaskSuspend( _
   ByVal taskId As Integer _
) As Boolean
```

[C#]

```
public bool TaskSuspend(
   int taskId
);
```

Parameters

taskId

An integer value that specifies the unique identifier associated with a background task.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskSuspend** method will suspend execution of the background worker thread associated with the task.

Once the task has been suspended, it will no longer be scheduled for execution, however the client session will remain active and the task may be resumed using the **TaskResume** method. Note that if a task is suspended for a long period of time, the background operation may fail because it has exceeded the timeout period imposed by the server.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.TaskSuspend Overload List

FileTransfer.TaskWait Method

Wait for all asynchronous tasks to complete.

Overload List

Wait for all asynchronous tasks to complete.

public bool TaskWait();

Wait for an asynchronous task to complete.

public bool TaskWait(int);

Wait for an asynchronous task to complete.

public bool TaskWait(int,int);

Wait for an asynchronous task to complete.

public bool TaskWait(int,int,ref ErrorCode);

Wait for an asynchronous task to complete.

public bool TaskWait(int,int,ref int,ref ErrorCode);

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.TaskWait Method ()

Wait for all asynchronous tasks to complete.

[Visual Basic]
```
Overloads Public Function TaskWait() As Boolean
```

[C#]
```
public bool TaskWait();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This overloaded version of the **TaskWait** method will cause the current working thread to block until all background tasks created by this instance of the class have completed. If there are no active background tasks, this method will return to the caller immediately.

You should not call this version of the method from the main UI thread. Windows messages will not be processed while this method is blocked waiting for the background tasks to complete, and this can cause your application to appear non-responsive to the end-user. If you have a GUI application and you need to determine if all tasks have completed, create a timer to periodically check the value of the **TaskCount** property. When it returns zero, there are no active background tasks executing.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.TaskWait Overload List

FileTransfer.TaskWait Method (Int32)

Wait for an asynchronous task to complete.

[Visual Basic]

```
Overloads Public Function TaskWait( _
   ByVal taskId As Integer _
) As Boolean
```

[C#]

```
public bool TaskWait(
   int taskId
);
```

Parameters

taskId

An integer value that specifies the unique identifier associated with a background task.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskWait** method waits for the specified task to complete. This method will cause the current working thread to block for an indefinite period of time until the task completes. If the specified task has already completed at the time this method is called, the method will return immediately without causing the current thread to block.

You should not call this overloaded version of the method from the main UI thread. Windows messages will not be processed while this method is blocked waiting for the background task to complete, and this can cause your application to appear non-responsive to the end-user. If you have a GUI application and you need to determine if a background task has finished, create a timer to periodically call the **TaskDone** method. When it returns **true** (indicating that the task has completed), you can safely call **TaskWait** to obtain the elapsed time and last error code without blocking the current thread.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.TaskWait Overload List

FileTransfer.TaskWait Method (Int32, Int32)

Wait for an asynchronous task to complete.

[Visual Basic]

```
Overloads Public Function TaskWait( _
   ByVal taskId As Integer, _
   ByVal timeWait As Integer _
) As Boolean
```

[C#]

```
public bool TaskWait(
   int taskId,
   int timeWait
);
```

Parameters

taskId

An integer value that specifies the unique identifier associated with a background task.

timeWait

An integer value that specifies the number of milliseconds to wait for the background task to complete.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskWait** method waits for the specified task to complete. This method will cause the current working thread to block until the task completes or the amount of time exceeds the number of milliseconds specified by the caller. If the *timeWait* parameter is zero, then this method will poll the status of the task and return immediately to the caller. If the specified task has already completed at the time this method is called, the method will return immediately without causing the current thread to block.

You should not call this method from the main UI thread with a long timeout period to wait for a background task to complete. Windows messages will not be processed while this method is blocked waiting for the background task to complete, and this can cause your application to appear non-responsive to the end-user. If you have a GUI application and you need to determine if a background task has finished, create a timer to periodically call the **TaskDone** method. When it returns **true** (indicating that the task has completed), you can safely call **TaskWait** to obtain the elapsed time and last error code without blocking the current thread.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.TaskWait Overload List

FileTransfer.TaskWait Method (Int32, Int32, ErrorCode)

Wait for an asynchronous task to complete.

[Visual Basic]

```
Overloads Public Function TaskWait( _
   ByVal taskId As Integer, _
   ByVal timeWait As Integer, _
   ByRef taskError As ErrorCode _
) As Boolean
```

[C#]

```
public bool TaskWait(
   int taskId,
   int timeWait,
   ref ErrorCode taskError
);
```

Parameters

taskId

An integer value that specifies the unique identifier associated with a background task.

timeWait

An integer value that specifies the number of milliseconds to wait for the background task to complete.

taskError

An ErrorCode value passed by reference that will contain the last error code set by the asynchronous task.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskWait** method waits for the specified task to complete. This method will cause the current working thread to block until the task completes or the amount of time exceeds the number of milliseconds specified by the caller. If the *timeWait* parameter is zero, then this method will poll the status of the task and return immediately to the caller.

If the specified task has already completed at the time this method is called, the method will return immediately without causing the current thread to block. The ***taskError*** parameter will contain the last error code value that was set by the worker thread before it terminated. If the ***taskError*** value is zero, that means that the background task was successful and no error occurred. A non-zero value will indicate that the background task has failed.

You should not call this method from the main UI thread with a long timeout period to wait for a background task to complete. Windows messages will not be processed while this method is blocked waiting for the background task to complete, and this can cause your application to appear non-responsive to the end-user. If you have a GUI application and you need to determine if a background task has finished, create a timer to periodically call the **TaskDone** method. When it returns **true** (indicating that the task has completed), you can safely call **TaskWait** to obtain the elapsed time and last error code without blocking the current thread.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.TaskWait Overload List

FileTransfer.TaskWait Method (Int32, Int32, Int32, ErrorCode)

Wait for an asynchronous task to complete.

[Visual Basic]

```
Overloads Public Function TaskWait( _
   ByVal taskId As Integer, _
   ByVal timeWait As Integer, _
   ByRef timeElapsed As Integer, _
   ByRef taskError As ErrorCode _
) As Boolean
```

[C#]

```
public bool TaskWait(
   int taskId,
   int timeWait,
   ref int timeElapsed,
   ref ErrorCode taskError
);
```

Parameters

taskId
: An integer value that specifies the unique identifier associated with a background task.

timeWait
: An integer value that specifies the number of milliseconds to wait for the background task to complete.

timeElapsed
: An integer value passed by reference that will contain the elapsed time for the task in milliseconds when the method returns.

taskError
: An ErrorCode value passed by reference that will contain the last error code set by the asynchronous task.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskWait** method waits for the specified task to complete. This method will cause the current working thread to block until the task completes or the amount of time exceeds the number of milliseconds specified by the caller. If the *timeWait* parameter is zero, then this method will poll the status of the task and return immediately to the caller.

If the specified task has already completed at the time this method is called, the method will return immediately without causing the current thread to block. The *timeElapsed* parameter contain the number of milliseconds that it took for the task to complete. The *taskError* parameter will contain the last error code value that was set by the worker thread before it terminated. If the *taskError* value is zero, that means that the background task was successful and no error occurred. A non-zero value will indicate that the background task has failed.

You should not call this method from the main UI thread with a long timeout period to wait for a background task to complete. Windows messages will not be processed while this method is blocked

waiting for the background task to complete, and this can cause your application to appear non-responsive to the end-user. If you have a GUI application and you need to determine if a background task has finished, create a timer to periodically call the **TaskDone** method. When it returns **true** (indicating that the task has completed), you can safely call **TaskWait** to obtain the elapsed time and last error code without blocking the current thread.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.TaskWait Overload List

FileTransfer.Uninitialize Method

Uninitialize the class library and release any resources allocated for the current thread.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method terminates any active connection, releases resources allocated for the current thread and unloads the networking library. After this method has been called, no further network operations may be performed until the class instance has been re-initialized.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the class is no longer needed.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.VerifyFile Method

Verify that the contents of a file on the local system are the same as the specified file on the server.

Overload List

Verify that the contents of a file on the local system are the same as the specified file on the server.

public bool VerifyFile(string,string);

Verify that the contents of a file on the local system are the same as the specified file on the server.

public bool VerifyFile(string,string,FileVerifyOptions);

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.VerifyFile Method (String, String)

Verify that the contents of a file on the local system are the same as the specified file on the server.

[Visual Basic]

```
Overloads Public Function VerifyFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String _
) As Boolean
```

[C#]

```
public bool VerifyFile(
   string localFile,
   string remoteFile
);
```

Parameters

localFile

A string value which specifies the name of the local file.

remoteFile

A string value which specifies the name of the file on the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **VerifyFile** method will attempt to verify that the contents of the local and remote files are identical using one of several methods, based on the features that the server supports. Preference will be given to the most reliable method available, using either an MD5 hash, a CRC32 checksum or comparing the size of the file, in that order.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.VerifyFile Overload List

FileTransfer.VerifyFile Method (String, String, FileVerifyOptions)

Verify that the contents of a file on the local system are the same as the specified file on the server.

[Visual Basic]

```
Overloads Public Function VerifyFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal options As FileVerifyOptions _
) As Boolean
```

[C#]

```
public bool VerifyFile(
   string localFile,
   string remoteFile,
   FileVerifyOptions options
);
```

Parameters

localFile

A string value which specifies the name of the local file.

remoteFile

A string value which specifies the name of the file on the server.

options

An FileVerifyOptions enumeration which specifies one or more options that should be used when comparing the files.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **VerifyFile** method will attempt to verify that the contents of the local and remote files are identical using one of several methods, based on the features that the server supports. Preference will be given to the most reliable method available, using either an MD5 hash, a CRC32 checksum or comparing the size of the file, in that order.

See Also

FileTransfer Class | SocketTools Namespace | FileTransfer.VerifyFile Overload List

FileTransfer Events

The events of the **FileTransfer** class are listed below. For a complete list of **FileTransfer** class members, see the FileTransfer Members topic.

Public Instance Events

⚡ OnCancel	Occurs when a blocking client operation is canceled.
⚡ OnCommand	Occurs when the client sends a command to the server and receives a reply indicating the result of that command.
⚡ OnError	Occurs when an network operation fails.
⚡ OnGetFile	Occurs when a file download has been initiated.
⚡ OnProgress	Occurs as a data stream is being read or written to the server.
⚡ OnPutFile	Occurs when a file upload is initiated.
⚡ OnTaskBegin	Occurs when an asynchronous task begins execution.
⚡ OnTaskEnd	Occurs when an asynchronous task completes.
⚡ OnTaskRun	Occurs while a background task is active.
⚡ OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.OnCancel Event

Occurs when a blocking client operation is canceled.

[Visual Basic]

```
Public Event OnCancel As EventHandler
```

[C#]

```
public event EventHandler OnCancel;
```

Remarks

The **OnCancel** event is generated when a blocking client operation, such as sending or receiving data, is canceled with the **Cancel** method. To assist in determining which operation was canceled, check the value of the **Status** property.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.OnCommand Event

Occurs when the client sends a command to the server and receives a reply indicating the result of that command.

[Visual Basic]
```
Public Event OnCommand As OnCommandEventHandler
```

[C#]
```
public event OnCommandEventHandler OnCommand;
```

Event Data

The event handler receives an argument of type FileTransfer.CommandEventArgs containing data related to this event. The following **FileTransfer.CommandEventArgs** properties provide information specific to this event.

Property	Description
ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.

Remarks

The **OnCommand** event is generated when the client receives a reply from the server after some action has been taken.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.CommandEventArgs Class

Provides data for the OnCommand event.

For a list of all members of this type, see FileTransfer.CommandEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FileTransfer.CommandEventArgs

[Visual Basic]

```
Public Class FileTransfer.CommandEventArgs
    Inherits EventArgs
```

[C#]

```
public class FileTransfer.CommandEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

CommandEventArgs specifies the result code and result string for the last command executed by the server.

The OnCommand event occurs whenever a command is executed on the server.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

FileTransfer.CommandEventArgs Members | SocketTools Namespace

FileTransfer.CommandEventArgs Members

FileTransfer.CommandEventArgs overview

Public Instance Constructors

FileTransfer.CommandEventArgs Constructor	Initializes a new instance of the FileTransfer.CommandEventArgs class.

Public Instance Properties

ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FileTransfer.CommandEventArgs Class | SocketTools Namespace

FileTransfer.CommandEventArgs Constructor

Initializes a new instance of the FileTransfer.CommandEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FileTransfer.CommandEventArgs();
```

See Also

FileTransfer.CommandEventArgs Class | SocketTools Namespace

FileTransfer.CommandEventArgs Properties

The properties of the **FileTransfer.CommandEventArgs** class are listed below. For a complete list of **FileTransfer.CommandEventArgs** class members, see the FileTransfer.CommandEventArgs Members topic.

Public Instance Properties

ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.

See Also

FileTransfer.CommandEventArgs Class | SocketTools Namespace

FileTransfer.CommandEventArgs.ResultCode Property

Gets a value which specifies the last result code returned by the server.

[Visual Basic]
```
Public ReadOnly Property ResultCode As Integer
```

[C#]
```
public int ResultCode {get;}
```

Property Value

An integer value which specifies the last result code returned by the server.

Remarks

This property should be checked after the **Command** method is used to execute a command on the server to determine if the operation was successful. Result codes are three-digit numeric values returned by the remote server and may be broken down into the following ranges:

ResultCode	Description
100-199	Positive preliminary result. This indicates that the requested action is being initiated, and the client should expect another reply from the server before proceeding.
200-299	Positive completion result. This indicates that the server has successfully completed the requested action.
300-399	Positive intermediate result. This indicates that the requested action cannot complete until additional information is provided to the server.
400-499	Transient negative completion result. This indicates that the requested action did not take place, but the error condition is temporary and may be attempted again.
500-599	Permanent negative completion result. This indicates that the requested action did not take place.

It is important to note that while some result codes have become standardized, not all servers respond to commands using the same result codes. For example, one server may respond with a result code of 221 to indicate success, while another may respond with a value of 235. It is recommended that applications check for ranges of values to determine if a command was successful, not a specific value.

See Also

FileTransfer.CommandEventArgs Class | SocketTools Namespace

FileTransfer.CommandEventArgs.ResultString Property

Gets a string value which describes the result of the previous command.

[Visual Basic]

```
Public ReadOnly Property ResultString As String
```

[C#]

```
public string ResultString {get;}
```

Property Value

A string which describes the result of the previous command executed on the server.

Remarks

This string is generated by the remote server, and typically is used to describe the result code. For example, if an error is indicated by the result code, the result string may describe the condition that caused the error.

See Also

FileTransfer.CommandEventArgs Class | SocketTools Namespace

FileTransfer.OnError Event

Occurs when an network operation fails.

[Visual Basic]
```
Public Event OnError As OnErrorEventHandler
```

[C#]
```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type FileTransfer.ErrorEventArgs containing data related to this event. The following **FileTransfer.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a client operation fails.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see FileTransfer.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FileTransfer.ErrorEventArgs

[Visual Basic]
```
Public Class FileTransfer.ErrorEventArgs
    Inherits EventArgs
```

[C#]
```
public class FileTransfer.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

FileTransfer.ErrorEventArgs Members | SocketTools Namespace

FileTransfer.ErrorEventArgs Members

FileTransfer.ErrorEventArgs overview

Public Instance Constructors

FileTransfer.ErrorEventArgs Constructor	Initializes a new instance of the FileTransfer.ErrorEventArgs class.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FileTransfer.ErrorEventArgs Class | SocketTools Namespace

FileTransfer.ErrorEventArgs Constructor

Initializes a new instance of the FileTransfer.ErrorEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FileTransfer.ErrorEventArgs();
```

See Also

FileTransfer.ErrorEventArgs Class | SocketTools Namespace

FileTransfer.ErrorEventArgs Properties

The properties of the **FileTransfer.ErrorEventArgs** class are listed below. For a complete list of **FileTransfer.ErrorEventArgs** class members, see the FileTransfer.ErrorEventArgs Members topic.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

FileTransfer.ErrorEventArgs Class | SocketTools Namespace

FileTransfer.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

FileTransfer.ErrorEventArgs Class | SocketTools Namespace | Error Property

FileTransfer.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public FileTransfer.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

FileTransfer.ErrorEventArgs Class | SocketTools Namespace | Description Property

FileTransfer.OnGetFile Event

Occurs when a file download has been initiated.

[Visual Basic]
```
Public Event OnGetFile As OnGetFileEventHandler
```

[C#]
```
public event OnGetFileEventHandler OnGetFile;
```

Event Data

The event handler receives an argument of type FileTransfer.GetFileEventArgs containing data related to this event. The following **FileTransfer.GetFileEventArgs** properties provide information specific to this event.

Property	Description
LocalFile	Gets a value which specifies the name of the local file.
RemoteFile	Gets a value which specifies the name of the remote file.

Remarks

The **OnGetFile** event is generated when a file transfer is initiated by calling the **GetFile** or **GetMultipleFiles** methods. This will be followed by one or more **OnProgress** events which will indicate the progress of the transfer. If multiple files are being downloaded, this event will fire for each file as it is transferred.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.GetFileEventArgs Class

Provides data for the OnGetFile event.

For a list of all members of this type, see FileTransfer.GetFileEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FileTransfer.GetFileEventArgs

[Visual Basic]

```
Public Class FileTransfer.GetFileEventArgs
    Inherits EventArgs
```

[C#]

```
public class FileTransfer.GetFileEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

GetFileEventArgs specifies information about the start of a file transfer from the server to the local system.

The OnGetFile event occurs when either the **GetFile** or **GetMultipleFiles** methods are called.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

FileTransfer.GetFileEventArgs Members | SocketTools Namespace

FileTransfer.GetFileEventArgs Members

FileTransfer.GetFileEventArgs overview

Public Instance Constructors

FileTransfer.GetFileEventArgs Constructor	Initializes a new instance of the FileTransfer.GetFileEventArgs class.

Public Instance Properties

LocalFile	Gets a value which specifies the name of the local file.
RemoteFile	Gets a value which specifies the name of the remote file.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FileTransfer.GetFileEventArgs Class | SocketTools Namespace

FileTransfer.GetFileEventArgs Constructor

Initializes a new instance of the FileTransfer.GetFileEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FileTransfer.GetFileEventArgs();
```

See Also

FileTransfer.GetFileEventArgs Class | SocketTools Namespace

FileTransfer.GetFileEventArgs Properties

The properties of the **FileTransfer.GetFileEventArgs** class are listed below. For a complete list of **FileTransfer.GetFileEventArgs** class members, see the FileTransfer.GetFileEventArgs Members topic.

Public Instance Properties

LocalFile	Gets a value which specifies the name of the local file.
RemoteFile	Gets a value which specifies the name of the remote file.

See Also

FileTransfer.GetFileEventArgs Class | SocketTools Namespace

FileTransfer.GetFileEventArgs.LocalFile Property

Gets a value which specifies the name of the local file.

[Visual Basic]

```
Public ReadOnly Property LocalFile As String
```

[C#]

```
public string LocalFile {get;}
```

Property Value

A string which specifies the name of the local file.

See Also

FileTransfer.GetFileEventArgs Class | SocketTools Namespace

FileTransfer.GetFileEventArgs.RemoteFile Property

Gets a value which specifies the name of the remote file.

[Visual Basic]

```
Public ReadOnly Property RemoteFile As String
```

[C#]

```
public string RemoteFile {get;}
```

Property Value

A string value which specifies the name of the remote file.

See Also

FileTransfer.GetFileEventArgs Class | SocketTools Namespace

FileTransfer.OnProgress Event

Occurs as a data stream is being read or written to the server.

[Visual Basic]
```
Public Event OnProgress As OnProgressEventHandler
```

[C#]
```
public event OnProgressEventHandler OnProgress;
```

Event Data

The event handler receives an argument of type FileTransfer.ProgressEventArgs containing data related to this event. The following **FileTransfer.ProgressEventArgs** properties provide information specific to this event.

Property	Description
BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
FileName	Gets a value which specifies a file name.
FileSize	Gets a value which specifies the size of the file being transferred.
Percent	Gets a value which specifies the percentage of data that has been read or written.

Remarks

The **OnProgress** event occurs as a data stream is being read or written to the server. If large amounts of data are being read or written, this event can be used to update a progress bar or other user-interface component to provide the user with some visual feedback on the progress of the operation.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.ProgressEventArgs Class

Provides data for the OnProgress event.

For a list of all members of this type, see FileTransfer.ProgressEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FileTransfer.ProgressEventArgs

[Visual Basic]

```
Public Class FileTransfer.ProgressEventArgs
    Inherits EventArgs
```

[C#]

```
public class FileTransfer.ProgressEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ProgressEventArgs specifies the number of bytes copied from the data stream, the total number of bytes in the data stream and a completion percentage.

The OnProgress event occurs as a data stream is being read or written to the server.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

FileTransfer.ProgressEventArgs Members | SocketTools Namespace

FileTransfer.ProgressEventArgs Members

FileTransfer.ProgressEventArgs overview

Public Instance Constructors

FileTransfer.ProgressEventArgs Constructor	Initializes a new instance of the FileTransfer.ProgressEventArgs class.

Public Instance Properties

BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
FileName	Gets a value which specifies a file name.
FileSize	Gets a value which specifies the size of the file being transferred.
Percent	Gets a value which specifies the percentage of data that has been read or written.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FileTransfer.ProgressEventArgs Class | SocketTools Namespace

FileTransfer.ProgressEventArgs Constructor

Initializes a new instance of the FileTransfer.ProgressEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public FileTransfer.ProgressEventArgs();
```

See Also

FileTransfer.ProgressEventArgs Class | SocketTools Namespace

FileTransfer.ProgressEventArgs Properties

The properties of the **FileTransfer.ProgressEventArgs** class are listed below. For a complete list of **FileTransfer.ProgressEventArgs** class members, see the FileTransfer.ProgressEventArgs Members topic.

Public Instance Properties

BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
FileName	Gets a value which specifies a file name.
FileSize	Gets a value which specifies the size of the file being transferred.
Percent	Gets a value which specifies the percentage of data that has been read or written.

See Also

FileTransfer.ProgressEventArgs Class | SocketTools Namespace

FileTransfer.ProgressEventArgs.BytesCopied Property

Gets a value which specifies the number of bytes of data that has been read or written.

[Visual Basic]

```
Public ReadOnly Property BytesCopied As Long
```

[C#]

```
public long BytesCopied {get;}
```

Property Value

An integer value which specifies the number of bytes of data.

Remarks

The **BytesCopied** property specifies the number of bytes that have been read from the server and stored in the local stream buffer, or written from the stream buffer to the server.

See Also

FileTransfer.ProgressEventArgs Class | SocketTools Namespace | FileSize Property | Percent Property

FileTransfer.ProgressEventArgs.FileName Property

Gets a value which specifies a file name.

[Visual Basic]

```
Public ReadOnly Property FileName As String
```

[C#]

```
public string FileName {get;}
```

Property Value

A string value which specifies the name of the file being transferred.

See Also

FileTransfer.ProgressEventArgs Class | SocketTools Namespace

FileTransfer.ProgressEventArgs.FileSize Property

Gets a value which specifies the size of the file being transferred.

[Visual Basic]

```
Public ReadOnly Property FileSize As Long
```

[C#]

```
public long FileSize {get;}
```

Property Value

An integer value which specifies the number of bytes of data.

Remarks

The **FileSize** property specifies the total amount of data being read from the server and written to the local file, or sent from the local system to the server. If the amount of data was unknown or unspecified at the time the method call was made, then this value will always be the same as the **BytesCopied** property.

See Also

FileTransfer.ProgressEventArgs Class | SocketTools Namespace | BytesCopied Property | Percent Property

FileTransfer.ProgressEventArgs.Percent Property

Gets a value which specifies the percentage of data that has been read or written.

[Visual Basic]

```
Public ReadOnly Property Percent As Integer
```

[C#]

```
public int Percent {get;}
```

Property Value

An integer value which specifies a percentage.

Remarks

The **Percent** property specifies the percentage of data that has been transmitted, expressed as an integer value between 0 and 100, inclusive. If the maximum size of the data stream was not specified by the caller, this value will always be 100.

See Also

FileTransfer.ProgressEventArgs Class | SocketTools Namespace | BytesCopied Property | FileSize Property

FileTransfer.OnPutFile Event

Occurs when a file upload is initiated.

[Visual Basic]
```
Public Event OnPutFile As OnPutFileEventHandler
```

[C#]
```
public event OnPutFileEventHandler OnPutFile;
```

Event Data

The event handler receives an argument of type FileTransfer.PutFileEventArgs containing data related to this event. The following **FileTransfer.PutFileEventArgs** properties provide information specific to this event.

Property	Description
LocalFile	Gets a value which specifies the name of the local file.
RemoteFile	Gets a value which specifies the name of the remote file.

Remarks

The **OnPutFile** event is generated when a file transfer is initiated by calling the **PutFile** or **PutMultipleFiles** methods. This will be followed by one or more **OnProgress** events which will indicate the progress of the transfer. If multiple files are being uploaded, this event will fire for each file as it is transferred.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.PutFileEventArgs Class

Provides data for the OnPutFile event.

For a list of all members of this type, see FileTransfer.PutFileEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FileTransfer.PutFileEventArgs

[Visual Basic]

```
Public Class FileTransfer.PutFileEventArgs
    Inherits EventArgs
```

[C#]

```
public class FileTransfer.PutFileEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

PutFileEventArgs specifies information about the start of a file transfer from the local system to the server.

The OnPutFile event occurs when either the **PutFile** or **PutMultipleFiles** methods are called.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

FileTransfer.PutFileEventArgs Members | SocketTools Namespace

FileTransfer.PutFileEventArgs Members

FileTransfer.PutFileEventArgs overview

Public Instance Constructors

FileTransfer.PutFileEventArgs Constructor	Initializes a new instance of the FileTransfer.PutFileEventArgs class.

Public Instance Properties

LocalFile	Gets a value which specifies the name of the local file.
RemoteFile	Gets a value which specifies the name of the remote file.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FileTransfer.PutFileEventArgs Class | SocketTools Namespace

FileTransfer.PutFileEventArgs Constructor

Initializes a new instance of the FileTransfer.PutFileEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FileTransfer.PutFileEventArgs();
```

See Also

FileTransfer.PutFileEventArgs Class | SocketTools Namespace

FileTransfer.PutFileEventArgs Properties

The properties of the **FileTransfer.PutFileEventArgs** class are listed below. For a complete list of **FileTransfer.PutFileEventArgs** class members, see the FileTransfer.PutFileEventArgs Members topic.

Public Instance Properties

LocalFile	Gets a value which specifies the name of the local file.
RemoteFile	Gets a value which specifies the name of the remote file.

See Also

FileTransfer.PutFileEventArgs Class | SocketTools Namespace

FileTransfer.PutFileEventArgs.LocalFile Property

Gets a value which specifies the name of the local file.

[Visual Basic]

```
Public ReadOnly Property LocalFile As String
```

[C#]

```
public string LocalFile {get;}
```

Property Value

A string which specifies the name of the local file.

See Also

FileTransfer.PutFileEventArgs Class | SocketTools Namespace

FileTransfer.PutFileEventArgs.RemoteFile Property

Gets a value which specifies the name of the remote file.

[Visual Basic]

```
Public ReadOnly Property RemoteFile As String
```

[C#]

```
public string RemoteFile {get;}
```

Property Value

A string value which specifies the name of the remote file.

See Also

FileTransfer.PutFileEventArgs Class | SocketTools Namespace

FileTransfer.OnTaskBegin Event

Occurs when an asynchronous task begins execution.

[Visual Basic]

```
Public Event OnTaskBegin As OnTaskBeginEventHandler
```

[C#]

```
public event OnTaskBeginEventHandler OnTaskBegin;
```

Event Data

The event handler receives an argument of type FileTransfer.TaskBeginEventArgs containing data related to this event. The following **FileTransfer.TaskBeginEventArgs** property provides information specific to this event.

Property	Description
TaskId	Get the unique task identifier associated with the event.

Remarks

The **OnTaskBegin** event occurs when a background task associated with an asynchronous file transfer begins executing. This event can be used in conjunction with the **OnTaskEnd** event to monitor one or more background tasks that are created to perform asynchronous file transfers.

This event and the related asynchronous task events are invoked from the context of the thread that is managing the background task, and not the thread that created the class instance. If a handler is implemented for this event, its code will be executing in a different thread than the main UI thread. You should never attempt to update your application's user interface directly from within this event handler. Instead, you must create a delegate and use the **Invoke** method to ensure that any changes to the user interface are done within the context of the main UI thread.

Because background tasks are managed in separate threads, this has the effect of making your application multi-threaded, even if you do not explicitly create any worker threads in your own code. If the code in your event handler modifies a public member variable or shared object, you must ensure that access to that object is synchronized. For example, if your event handler updates a shared instance of a **Hashtable** object, you should ensure that all operations are performed through the thread-safe wrapper returned by the **Synchronized** method for that class. Refer to the MSDN documentation for more information about creating thread-safe applications.

See Also

FileTransfer Class | SocketTools Namespace | OnTaskEnd Event | OnTaskRun Event

FileTransfer.TaskBeginEventArgs Class

Provides data for the OnTaskBegin event.

For a list of all members of this type, see FileTransfer.TaskBeginEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FileTransfer.TaskBeginEventArgs

[Visual Basic]
```
Public Class FileTransfer.TaskBeginEventArgs
    Inherits EventArgs
```

[C#]
```
public class FileTransfer.TaskBeginEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

FileTransfer.TaskBeginEventArgs Members | SocketTools Namespace

FileTransfer.TaskBeginEventArgs Constructor

Initializes a new instance of the FileTransfer.TaskBeginEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public FileTransfer.TaskBeginEventArgs();
```

See Also

FileTransfer.TaskBeginEventArgs Class | SocketTools Namespace

FileTransfer.TaskBeginEventArgs Members

FileTransfer.TaskBeginEventArgs overview

Public Instance Constructors

FileTransfer.TaskBeginEventArgs Constructor	Initializes a new instance of the FileTransfer.TaskBeginEventArgs class.

Public Instance Properties

TaskId	Get the unique task identifier associated with the event.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FileTransfer.TaskBeginEventArgs Class | SocketTools Namespace

FileTransfer.TaskBeginEventArgs Properties

The properties of the **FileTransfer.TaskBeginEventArgs** class are listed below. For a complete list of **FileTransfer.TaskBeginEventArgs** class members, see the FileTransfer.TaskBeginEventArgs Members topic.

Public Instance Properties

TaskId	Get the unique task identifier associated with the event.

See Also

FileTransfer.TaskBeginEventArgs Class | SocketTools Namespace

FileTransfer.TaskBeginEventArgs.TaskId Property

Get the unique task identifier associated with the event.

[Visual Basic]

```
Public ReadOnly Property TaskId As Integer
```

[C#]

```
public int TaskId {get;}
```

Property Value

An integer value that uniquely identifies the task that invoked the event handler.

See Also

FileTransfer.TaskBeginEventArgs Class | SocketTools Namespace

FileTransfer.OnTaskEnd Event

Occurs when an asynchronous task completes.

[Visual Basic]
```
Public Event OnTaskEnd As OnTaskEndEventHandler
```

[C#]
```
public event OnTaskEndEventHandler OnTaskEnd;
```

Event Data

The event handler receives an argument of type FileTransfer.TaskEndEventArgs containing data related to this event. The following **FileTransfer.TaskEndEventArgs** properties provide information specific to this event.

Property	Description
Error	Get the last error code for the background task.
TaskId	Get the unique task identifier associated with the event.
TimeElapsed	Gets the amount of time that has elapsed in milliseconds.

Remarks

The **OnTaskEnd** event occurs when a file transfer completes and the background task has terminated. Refer to the **OnTaskBegin** event for additional information about implementing a handler for this event.

See Also

FileTransfer Class | SocketTools Namespace | OnTaskBegin Event | OnTaskRun Event

FileTransfer.TaskEndEventArgs Class

Provides data for the OnTaskEnd event.

For a list of all members of this type, see FileTransfer.TaskEndEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FileTransfer.TaskEndEventArgs

[Visual Basic]

```
Public Class FileTransfer.TaskEndEventArgs
    Inherits EventArgs
```

[C#]

```
public class FileTransfer.TaskEndEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

FileTransfer.TaskEndEventArgs Members | SocketTools Namespace

FileTransfer.TaskEndEventArgs Constructor

Initializes a new instance of the FileTransfer.TaskEndEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public FileTransfer.TaskEndEventArgs();
```

See Also

FileTransfer.TaskEndEventArgs Class | SocketTools Namespace

FileTransfer.TaskEndEventArgs Members

FileTransfer.TaskEndEventArgs overview

Public Instance Constructors

FileTransfer.TaskEndEventArgs Constructor	Initializes a new instance of the FileTransfer.TaskEndEventArgs class.

Public Instance Properties

Error	Get the last error code for the background task.
TaskId	Get the unique task identifier associated with the event.
TimeElapsed	Gets the amount of time that has elapsed in milliseconds.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FileTransfer.TaskEndEventArgs Class | SocketTools Namespace

FileTransfer.TaskEndEventArgs Properties

The properties of the **FileTransfer.TaskEndEventArgs** class are listed below. For a complete list of **FileTransfer.TaskEndEventArgs** class members, see the FileTransfer.TaskEndEventArgs Members topic.

Public Instance Properties

Error	Get the last error code for the background task.
TaskId	Get the unique task identifier associated with the event.
TimeElapsed	Gets the amount of time that has elapsed in milliseconds.

See Also

FileTransfer.TaskEndEventArgs Class | SocketTools Namespace

FileTransfer.TaskEndEventArgs.Error Property

Get the last error code for the background task.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public FileTransfer.ErrorCode Error {get;}
```

Property Value

An **ErrorCode** enumeration that specifies the last error code set by the background task.

See Also

FileTransfer.TaskEndEventArgs Class | SocketTools Namespace

FileTransfer.TaskEndEventArgs.TaskId Property

Get the unique task identifier associated with the event.

[Visual Basic]

```
Public ReadOnly Property TaskId As Integer
```

[C#]

```
public int TaskId {get;}
```

Property Value

An integer value that uniquely identifies the task that invoked the event handler.

See Also

FileTransfer.TaskEndEventArgs Class | SocketTools Namespace

FileTransfer.TaskEndEventArgs.TimeElapsed Property

Gets the amount of time that has elapsed in milliseconds.

[Visual Basic]

```
Public ReadOnly Property TimeElapsed As Integer
```

[C#]

```
public int TimeElapsed {get;}
```

Property Value

An integer value that specifies the number of milliseconds that the background task has executed.

See Also

FileTransfer.TaskEndEventArgs Class | SocketTools Namespace

FileTransfer.OnTaskRun Event

Occurs while a background task is active.

```
[Visual Basic]
Public Event OnTaskRun As OnTaskRunEventHandler
```

```
[C#]
public event OnTaskRunEventHandler OnTaskRun;
```

Event Data

The event handler receives an argument of type FileTransfer.TaskRunEventArgs containing data related to this event. The following **FileTransfer.TaskRunEventArgs** properties provide information specific to this event.

Property	Description
Completed	Gets an estimate of the progress of the background task.
TaskId	Get the unique task identifier associated with the event.
TimeElapsed	Gets the amount of time that has elapsed in milliseconds.

Remarks

The **OnTaskRun** event is generated periodically during a file transfer while the background task is active. The rate and number of times that this event will be generated depends on the task being performed. This event is generally analogous to the **OnProgress** event for file transfers that are performed in the current working thread, however the **OnTaskRun** event will occur for each individual background task that is active.

Refer to the **OnTaskBegin** event for additional information about implementing a handler for this event.

See Also

FileTransfer Class | SocketTools Namespace | OnTaskBegin Event | OnTaskEnd Event

FileTransfer.TaskRunEventArgs Class

Provides data for the OnTaskRun event.

For a list of all members of this type, see FileTransfer.TaskRunEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FileTransfer.TaskRunEventArgs

[Visual Basic]

```
Public Class FileTransfer.TaskRunEventArgs
    Inherits EventArgs
```

[C#]

```
public class FileTransfer.TaskRunEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

FileTransfer.TaskRunEventArgs Members | SocketTools Namespace

FileTransfer.TaskRunEventArgs Constructor

Initializes a new instance of the FileTransfer.TaskRunEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FileTransfer.TaskRunEventArgs();
```

See Also

FileTransfer.TaskRunEventArgs Class | SocketTools Namespace

FileTransfer.TaskRunEventArgs Members

FileTransfer.TaskRunEventArgs overview

Public Instance Constructors

FileTransfer.TaskRunEventArgs Constructor	Initializes a new instance of the FileTransfer.TaskRunEventArgs class.

Public Instance Properties

Completed	Gets an estimate of the progress of the background task.
TaskId	Get the unique task identifier associated with the event.
TimeElapsed	Gets the amount of time that has elapsed in milliseconds.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FileTransfer.TaskRunEventArgs Class | SocketTools Namespace

FileTransfer.TaskRunEventArgs Properties

The properties of the **FileTransfer.TaskRunEventArgs** class are listed below. For a complete list of **FileTransfer.TaskRunEventArgs** class members, see the FileTransfer.TaskRunEventArgs Members topic.

Public Instance Properties

Completed	Gets an estimate of the progress of the background task.
TaskId	Get the unique task identifier associated with the event.
TimeElapsed	Gets the amount of time that has elapsed in milliseconds.

See Also

FileTransfer.TaskRunEventArgs Class | SocketTools Namespace

FileTransfer.TaskRunEventArgs.Completed Property

Gets an estimate of the progress of the background task.

[Visual Basic]

```
Public ReadOnly Property Completed As Integer
```

[C#]

```
public int Completed {get;}
```

Property Value

An integer value that returns a number between 0 and 100 inclusive that specifies the estimated percentage of completion for the task. A value of zero indicates that the task has just begun executing, while a value of 100 indicates that the task is at or near completion.

See Also

FileTransfer.TaskRunEventArgs Class | SocketTools Namespace

FileTransfer.TaskRunEventArgs.TaskId Property

Get the unique task identifier associated with the event.

[Visual Basic]

```
Public ReadOnly Property TaskId As Integer
```

[C#]

```
public int TaskId {get;}
```

Property Value

An integer value that uniquely identifies the task that invoked the event handler.

See Also

FileTransfer.TaskRunEventArgs Class | SocketTools Namespace

FileTransfer.TaskRunEventArgs.TimeElapsed Property

Gets the amount of time that has elapsed in milliseconds.

[Visual Basic]

```
Public ReadOnly Property TimeElapsed As Integer
```

[C#]

```
public int TimeElapsed {get;}
```

Property Value

An integer value that specifies the number of milliseconds that the background task has been executing.

See Also

FileTransfer.TaskRunEventArgs Class | SocketTools Namespace

FileTransfer.OnTimeout Event

Occurs when a blocking operation fails to complete before the timeout period elapses.

[Visual Basic]

```
Public Event OnTimeout As EventHandler
```

[C#]

```
public event EventHandler OnTimeout;
```

Remarks

The **OnTimeout** event occurs when a blocking operation, such as sending or receiving data on the client, fails to complete before the specified timeout period elapses. The timeout period for a blocking operation can be adjusted by setting the **Timeout** property.

See Also

FileTransfer Class | SocketTools Namespace

FileTransfer.FileInformation Structure

This structure is used by the GetFirstFile and GetNextFile methods to return information about a file on the server.

For a list of all members of this type, see FileTransfer.FileInformation Members.

System.Object
  System.ValueType
    SocketTools.FileTransfer.FileInformation

[Visual Basic]
```
Public Structure FileTransfer.FileInformation
```

[C#]
```
public struct FileTransfer.FileInformation
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

FileTransfer.FileInformation Members | SocketTools Namespace

FileTransfer.FileInformation Members

FileTransfer.FileInformation overview

Public Instance Fields

FileDate	Specifies the date and time the file was created or last modified.
FileGroup	Specifies the name of the group that owns the file.
FileLinks	Specifies the number of links to the file.
FileName	Specifies the name of the file.
FileOwner	Specifies the name of the file owner.
FilePerms	Specifies the permissions for the file.
FileSize	Specifies the size of the file in bytes.
FileVersion	Specifies the number of revisions made to the file.
IsDirectory	Specifies if the file is a directory or regular file.

Public Instance Methods

Equals (inherited from ValueType)	Indicates whether this instance and a specified object are equal.
GetHashCode (inherited from ValueType)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from ValueType)	Returns the fully qualified type name of this instance.

See Also

FileTransfer.FileInformation Class | SocketTools Namespace

FileTransfer.FileInformation Fields

The fields of the **FileTransfer.FileInformation** structure are listed below. For a complete list of **FileTransfer.FileInformation** structure members, see the FileTransfer.FileInformation Members topic.

Public Instance Fields

FileDate	Specifies the date and time the file was created or last modified.
FileGroup	Specifies the name of the group that owns the file.
FileLinks	Specifies the number of links to the file.
FileName	Specifies the name of the file.
FileOwner	Specifies the name of the file owner.
FilePerms	Specifies the permissions for the file.
FileSize	Specifies the size of the file in bytes.
FileVersion	Specifies the number of revisions made to the file.
IsDirectory	Specifies if the file is a directory or regular file.

See Also

FileTransfer.FileInformation Class | SocketTools Namespace

FileTransfer.FileInformation.FileDate Field

Specifies the date and time the file was created or last modified.

[Visual Basic]
```
Public FileDate As Date
```

[C#]
```
public DateTime FileDate;
```

Remarks

Some file servers may not return a specific time value if the file was last created or modified more than six months ago. In that case, only the date will be returned.

See Also

FileTransfer.FileInformation Class | SocketTools Namespace

FileTransfer.FileInformation.FileGroup Field

Specifies the name of the group that owns the file.

[Visual Basic]

```
Public FileGroup As String
```

[C#]

```
public string FileGroup;
```

See Also

FileTransfer.FileInformation Class | SocketTools Namespace

FileTransfer.FileInformation.FileLinks Field

Specifies the number of links to the file.

[Visual Basic]
```
Public FileLinks As Integer
```

[C#]
```
public int FileLinks;
```

See Also

FileTransfer.FileInformation Class | SocketTools Namespace

FileTransfer.FileInformation.FileName Field

Specifies the name of the file.

[Visual Basic]

```
Public FileName As String
```

[C#]

```
public string FileName;
```

See Also

FileTransfer.FileInformation Class | SocketTools Namespace

FileTransfer.FileInformation.FileOwner Field

Specifies the name of the file owner.

[Visual Basic]

```
Public FileOwner As String
```

[C#]

```
public string FileOwner;
```

See Also

FileTransfer.FileInformation Class | SocketTools Namespace

FileTransfer.FileInformation.FilePerms Field

Specifies the permissions for the file.

[Visual Basic]
```
Public FilePerms As FilePermissions
```

[C#]
```
public FilePermissions FilePerms;
```

Remarks

For those familiar with UNIX, the file permissions are the same as those used by the **chmod** command. For the proprietary Sterling directory formats, a bit map representing the status codes and transfer protocol of the file are stored in this member.

See Also

FileTransfer.FileInformation Class | SocketTools Namespace

FileTransfer.FileInformation.FileSize Field

Specifies the size of the file in bytes.

[Visual Basic]

```
Public FileSize As Long
```

[C#]

```
public long FileSize;
```

See Also

FileTransfer.FileInformation Class | SocketTools Namespace

FileTransfer.FileInformation.FileVersion Field

Specifies the number of revisions made to the file.

[Visual Basic]

```
Public FileVersion As Integer
```

[C#]

```
public int FileVersion;
```

See Also

FileTransfer.FileInformation Class | SocketTools Namespace

FileTransfer.FileInformation.IsDirectory Field

Specifies if the file is a directory or regular file.

[Visual Basic]

```
Public IsDirectory As Boolean
```

[C#]

```
public bool IsDirectory;
```

See Also

FileTransfer.FileInformation Class | SocketTools Namespace

FileTransfer.PortRange Structure

This structure is used by the ActivePorts property which allows the client to specify the port range used for active mode file transfers.

For a list of all members of this type, see FileTransfer.PortRange Members.

System.Object
  System.ValueType
    SocketTools.FileTransfer.PortRange

[Visual Basic]
```
Public Structure FileTransfer.PortRange
```

[C#]
```
public struct FileTransfer.PortRange
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

FileTransfer.PortRange Members | SocketTools Namespace

FileTransfer.PortRange Members

FileTransfer.PortRange overview

Public Instance Constructors

FileTransfer.PortRange Constructor	Initializes a new instance of the **PortRange** structure with a specified range of port numbers.

Public Instance Fields

HighPort	Specifies the high port number used for active file transfers.
LowPort	Specifies the low port number used for active file transfers.

Public Instance Methods

Equals (inherited from ValueType)	Indicates whether this instance and a specified object are equal.
GetHashCode (inherited from ValueType)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from ValueType)	Returns the fully qualified type name of this instance.

See Also

FileTransfer.PortRange Class | SocketTools Namespace

FileTransfer.PortRange Constructor

Initializes a new instance of the **PortRange** structure with a specified range of port numbers.

[Visual Basic]

```
Public Sub New( _
   ByVal lowPort As Integer, _
   ByVal highPort As Integer _
)
```

[C#]

```
public FileTransfer.PortRange(
   int lowPort,
   int highPort
);
```

Parameters

lowPort

An integer value which specifies the low port number.

highPort

An integer value which specifies the high port number.

Return Value

The minimum port number is 1025 and the maximum port number is 65535. Port numbers outside of this range of values will be silently adjusted.

See Also

FileTransfer.PortRange Class | SocketTools Namespace

FileTransfer.PortRange Fields

The fields of the **FileTransfer.PortRange** structure are listed below. For a complete list of **FileTransfer.PortRange** structure members, see the FileTransfer.PortRange Members topic.

Public Instance Fields

HighPort	Specifies the high port number used for active file transfers.
LowPort	Specifies the low port number used for active file transfers.

See Also

FileTransfer.PortRange Class | SocketTools Namespace

FileTransfer.PortRange.HighPort Field

Specifies the high port number used for active file transfers.

[Visual Basic]

```
Public HighPort As Integer
```

[C#]

```
public int HighPort;
```

Remarks

While this member may be assigned to any integer value, the port range is normalized and the maximum port number that may be used is 65535.

See Also

FileTransfer.PortRange Class | SocketTools Namespace

FileTransfer.PortRange.LowPort Field

Specifies the low port number used for active file transfers.

[Visual Basic]

```
Public LowPort As Integer
```

[C#]

```
public int LowPort;
```

Remarks

While this member may be assigned to any integer value, the port range is normalized and the minimum port number that may be used is 1025.

See Also

FileTransfer.PortRange Class | SocketTools Namespace

FileTransfer.SecureChannel Structure

The structure used to modify or return the current mode for the secure command or data channel.

For a list of all members of this type, see FileTransfer.SecureChannel Members.

System.Object
  System.ValueType
    SocketTools.FileTransfer.SecureChannel

[Visual Basic]
```
Public Structure FileTransfer.SecureChannel
```

[C#]
```
public struct FileTransfer.SecureChannel
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The **SecureChannel** structure specifies the current channel modes for the client session. This structure is used with the **ChannelMode** property when a secure connection is established with the server.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

FileTransfer.SecureChannel Members | SocketTools Namespace

FileTransfer.SecureChannel Members

FileTransfer.SecureChannel overview

Public Instance Constructors

FileTransfer.SecureChannel Constructor	Initializes a new instance of the **SecureChannel** structure with the specified values.

Public Instance Fields

Command	Specifies the secure channel mode used when sending commands to the server.
Data	Specifies the secure channel mode used when uploading or downloading files.

Public Instance Methods

Equals (inherited from ValueType)	Indicates whether this instance and a specified object are equal.
GetHashCode (inherited from ValueType)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from ValueType)	Returns the fully qualified type name of this instance.

See Also

FileTransfer.SecureChannel Class | SocketTools Namespace

FileTransfer.SecureChannel Constructor

Initializes a new instance of the **SecureChannel** structure with the specified values.

[Visual Basic]

```
Public Sub New( _
   ByVal commandMode As FtpChannelMode, _
   ByVal dataMode As FtpChannelMode _
)
```

[C#]

```
public FileTransfer.SecureChannel(
   FtpChannelMode commandMode,
   FtpChannelMode dataMode
);
```

Parameters

commandMode

A FtpChannelMode enumeration which specifies the channel mode used for commands sent to the server.

dataMode

A FtpChannelMode enumeration which specifies the channel mode used for file transfers.

See Also

FileTransfer.SecureChannel Class | SocketTools Namespace

FileTransfer.SecureChannel Fields

The fields of the **FileTransfer.SecureChannel** structure are listed below. For a complete list of **FileTransfer.SecureChannel** structure members, see the FileTransfer.SecureChannel Members topic.

Public Instance Fields

◆ Command	Specifies the secure channel mode used when sending commands to the server.
◆ Data	Specifies the secure channel mode used when uploading or downloading files.

See Also

FileTransfer.SecureChannel Class | SocketTools Namespace

FileTransfer.SecureChannel.Command Field

Specifies the secure channel mode used when sending commands to the server.

[Visual Basic]

```
Public Command As FtpChannelMode
```

[C#]

```
public FtpChannelMode Command;
```

See Also

FileTransfer.SecureChannel Class | SocketTools Namespace

FileTransfer.SecureChannel.Data Field

Specifies the secure channel mode used when uploading or downloading files.

[Visual Basic]
```
Public Data As FtpChannelMode
```

[C#]
```
public FtpChannelMode Data;
```

See Also

FileTransfer.SecureChannel Class | SocketTools Namespace

FileTransfer.OnCommandEventHandler Delegate

Represents the method that will handle the OnCommand event.

[Visual Basic]

```
Public Delegate Sub FileTransfer.OnCommandEventHandler( _
   ByVal sender As Object, _
   ByVal e As CommandEventArgs _
)
```

[C#]

```
public delegate void FileTransfer.OnCommandEventHandler(
      object sender,
      CommandEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A CommandEventArgs object that contains the event data.

Remarks

When you create an **OnCommandEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnCommandEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace

FileTransfer.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub FileTransfer.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void FileTransfer.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender
The source of the event.

e
An ErrorEventArgs object that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace

FileTransfer.OnGetFileEventHandler Delegate

Represents the method that will handle the OnGetFile event.

[Visual Basic]

```
Public Delegate Sub FileTransfer.OnGetFileEventHandler( _
   ByVal sender As Object, _
   ByVal e As GetFileEventArgs _
)
```

[C#]

```
public delegate void FileTransfer.OnGetFileEventHandler(
      object sender,
      GetFileEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A GetFileEventArgs object which contains the event data.

Remarks

When you create an **OnGetFileEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnGetFileEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace

FileTransfer.OnProgressEventHandler Delegate

Represents the method that will handle the OnProgress event.

[Visual Basic]

```
Public Delegate Sub FileTransfer.OnProgressEventHandler( _
   ByVal sender As Object, _
   ByVal e As ProgressEventArgs _
)
```

[C#]

```
public delegate void FileTransfer.OnProgressEventHandler(
      object sender,
      ProgressEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A ProgressEventArgs object that contains the event data.

Remarks

When you create an **OnProgressEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnProgressEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace

FileTransfer.OnPutFileEventHandler Delegate

Represents the method that will handle the OnPutFile event.

[Visual Basic]

```
Public Delegate Sub FileTransfer.OnPutFileEventHandler( _
   ByVal sender As Object, _
   ByVal e As PutFileEventArgs _
)
```

[C#]

```
public delegate void FileTransfer.OnPutFileEventHandler(
      object sender,
      PutFileEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A PutFileEventArgs object that contains the event data.

Remarks

When you create an **OnPutFileEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnPutFileEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace

FileTransfer.FilePermissions Enumeration

Specifies the access permissions for a file on the server.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum FileTransfer.FilePermissions
```

```
[C#]
[Flags]
public enum FileTransfer.FilePermissions
```

Members

Member Name	Description	Value
symbolicLink	The file is a symbolic link to another file. Symbolic links are special types of files found on UNIX based systems which are similar to Windows shortcuts.	4096
ownerRead	The owner has permission to open the file for reading. If the current user is the owner of the file, this grants the user the right to download the file to the local system.	1024
ownerWrite	The owner has permission to open the file for writing. If the current user is the owner of the file, this grants the user the right to replace the file. If this permission is set for a directory, this grants the user the right to create and delete files.	512
ownerExecute	The owner has permission to execute the contents of the file. The file is typically either a binary executable, script or batch file. If this permission is set for a directory, this may also grant the user the right to open that directory and search for files in that directory.	256
groupRead	Users in the specified group have permission to open the file for reading. If the current user is in the same group as the file owner, this grants the user the right to download the file.	64
groupWrite	Users in the specified group have permission to open the file for writing. On some platforms, this may also imply permission to delete the file. If the current user is in the same group as the	32

	file owner, this grants the user the right to replace the file. If this permission is set for a directory, this grants the user the right to create and delete files.	
groupExecute	Users in the specified group have permission to execute the contents of the file. If this permission is set for a directory, this may also grant the user the right to open that directory and search for files in that directory.	16
worldRead	All users have permission to open the file for reading. This permission grants any user the right to download the file to the local system.	4
worldWrite	All users have permission to open the file for writing. This permission grants any user the right to replace the file. If this permission is set for a directory, this grants any user the right to create and delete files.	2
worldExecute	All users have permission to execute the contents of the file. If this permission is set for a directory, this may also grant all users the right to open that directory and search for files in that directory.	1

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace

FileTransfer.ErrorCode Enumeration

Specifies the error codes returned by the FileTransfer class.

[Visual Basic]
```
Public Enum FileTransfer.ErrorCode
```

[C#]
```
public enum FileTransfer.ErrorCode
```

Remarks

The FileTransfer class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.
errorProxyAuthenticationRequired	Proxy authentication required.

errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.
errorUnsupportedMediaType	Request specified an unsupported media type.

errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.
errorNoMoreMailboxes	No more mailboxes exist on this server.

errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.
errorInvalidThreadHandle	The specified thread handle is no longer valid.

errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been defined.

errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or unrecognized.

errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

FileTransfer.FileServerType Enumeration

Specifies the type of server that the client is connected to.

[Visual Basic]

```
Public Enum FileTransfer.FileServerType
```

[C#]

```
public enum FileTransfer.FileServerType
```

Members

Member Name	Description
serverUnknown	The server type has not been explicitly set. The server type will be automatically determined by the value of the remote file URL or the service port number specified by the value of the **ServerPort** property.
serverFtp	The File Transfer Protocol is used when establishing a connection to the server.
serverHttp	The Hypertext Transfer Protocol is used when establishing a connection to the server.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace | ServerType Property (SocketTools.FileTransfer)

FileTransfer.FileTransferOptions Enumeration

Specifies the options that the FileTransfer class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum FileTransfer.FileTransferOptions
```

[C#]

```
[Flags]
public enum FileTransfer.FileTransferOptions
```

Remarks

The FileTransfer class uses the **FileTransferOptions** enumeration to specify one or more options to be used when establishing a connection with a remote host. Multiple options may be specified if necessary.

Members

Member Name	Description	Value
optionNone	No option specified.	0
optionDefault	The default connection option. This is the same as specifying **optionPassive**.	1
optionPassive	This option specifies the client should attempt to establish a passive connection to the server. This means that instead of the client opening a port on the local system and waiting for the server to establish a connection back to the client, the client will establish a second data connection to the server. This mode is recommended for most systems that are behind a NAT router or firewall.	1
optionSecure	This option specifies the client should attempt to establish a secure connection with the server. Note that the server must support secure connections using either the SSL or TLS protocol. This option specifies the client should attempt to establish a secure connection with the server. Note that the server must support secure connections using either the SSL or TLS protocol.	4096
optionImplicitSSL	This option specifies the client should attempt to establish a secure implicit SSL session. The SSL handshake is initiated immediately after the	4096

	connection to the server has been established.	
optionExplicitSSL	This option specifies the client should attempt to establish a secure explicit SSL session. The initial connection to the server is not encrypted, and the client will attempt to negotiate a secure connection by sending a command to the server. Some servers may require this option when connecting to the server on ports other than the default secure port of 990.	8192
optionSecureShell	This option specifies the client should attempt to establish a secure connection using the Secure Shell (SSH) protocol. This option is automatically selected if the connection is established on port 22, the standard port for SSH connections. It is only necessary to specify this option if the SSH connection must be established on a non-standard port.	16384
optionSecureFallback	This option specifies the client should permit the use of less secure cipher suites for compatibility with legacy servers. If this option is specified, the client will permit connections using TLS 1.0 and cipher suites that use RC4, MD5 and SHA1.	32768
optionFreeThread	This option specifies that class methods may be called from any thread, and not only the thread that established the connection. Using this option disables certain internal safety checks that are made by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads.	524288
optionHiResTimer	This option specifies that elapsed time values should be returned in milliseconds rather than seconds. This option is intended to provide greater accuracy with smaller file transfers over a high speed network connection.	1048576
optionTLSReuse	This option specifies that TLS session reuse should be enabled when establishing a secure FTP connection. This option is only supported on Windows 8.1 or Windows Server 2012	2097152

	R2 and later platforms, and it should only be used when explicitly required by the server. This option is not compatible with servers built using OpenSSL 1.0.2 and earlier versions which do not provide Extended Master Secret (EMS) support as outlined in RFC7627.	

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace

FileTransfer.FileTransferPriority Enumeration

Specifies the transfer priorities that the FileTransfer class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum FileTransfer.FileTransferPriority
```

[C#]

```
[Flags]
public enum FileTransfer.FileTransferPriority
```

Members

Member Name	Description	Value
priorityBackground	This priority significantly reduces the memory, processor and network resource utilization for the transfer. It is typically used with worker threads running in the background when the amount of time required perform the transfer is not critical.	0
priorityLow	This priority lowers the overall resource utilization for the transfer and meters the bandwidth allocated for the transfer. This priority will increase the average amount of time required to complete a file transfer.	1
priorityNormal	The default priority which balances resource utilization and transfer speed. It is recommended that most applications use this priority.	2
priorityHigh	This priority increases the overall resource utilization for the transfer, allocating more memory for internal buffering. It can be used when it is important to transfer the file quickly, and there are no other threads currently performing file transfers at the time.	3
priorityCritical	This priority can significantly increase processor, memory and network utilization while attempting to transfer the file as quickly as possible. If the file transfer is being performed in the main UI thread, this priority can cause the application to appear to become non-responsive. No events will be generated during the transfer.	4

priorityInvalid	An invalid transfer priority which indicates an error condition.	-1
priorityDefault	The default transfer priority. This is the same as specifying **priorityNormal**.	2
priorityLowest	The lowest valid transfer priority. This is the same as specifying **priorityBackground**.	0
priorityHighest	The highest valid transfer priority. This is the same as specifying **priorityCritical**.	4

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace

FileTransfer.FileTransferProxy Enumeration

Specifies the type of proxy server that is being used by the FileTransfer class.

[Visual Basic]
```
Public Enum FileTransfer.FileTransferProxy
```

[C#]
```
public enum FileTransfer.FileTransferProxy
```

Members

Member Name	Description
proxyNone	No proxy server is being used. This is the default value.
proxyFtpUser	The client is not logged into the proxy server. The USER command is sent in the format username@ftpsite followed by the password. This is the format used with the Gauntlet proxy server.
proxyFtpLogin	The client is logged into the FTP proxy server. The USER command is then sent in the format username@ftpsite followed by the password. This is the format used by the InterLock proxy server.
proxyFtpOpen	The client is not logged into the FTP proxy server. The OPEN command is sent specifying the host name, followed by the username and password.
proxyFtpSite	The client is logged into the FTP server. The SITE command is sent, specifying the host name, followed by the username and the password.
proxyFtpOther	This special proxy type specifies that another, undefined FTP proxy server is being used. The client connects to the proxy host, but does not attempt to authenticate the client. The application is responsible for negotiating with the proxy server, typically using the Command property to send specific command sequences.
proxyHttpStandard	A standard HTTP connection is established through the specified proxy server, and all resource requests will be specified using a complete URL. This proxy type should be used with standard HTTP connections.
proxyHttpSecure	A secure HTTP connection is established through the specified proxy server. This proxy type should only be used with secure connections and the Secure property should also be set to a value of true.
proxyHttpWindows	The configuration options for the current system should be used. These options are found in the

	Network and Internet settings for Windows.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace

FileTransfer.FileTransferType Enumeration

Specifies the type of file being transferred.

[Visual Basic]
```
Public Enum FileTransfer.FileTransferType
```

[C#]
```
public enum FileTransfer.FileTransferType
```

Members

Member Name	Description
fileAuto	The file type should be automatically determined based on the file name extension. If the file extension is unknown, the file type should be determined based on the contents of the file. The class has an internal list of common text file extensions, and additional file extensions can be registered using the **AddFileType** method.
fileAscii	The file being transferred is an ASCII text file. The characters the mark the end of a line (for example, a carriage return/linefeed pair under MS-DOS) are automatically converted to the format used by the target operating system.
fileEbcdic	The file being transferred is a text file created using the EBCDIC character set. If a file is being copied to a remote system, the ASCII characters are automatically converted to EBCDIC. If the file is being retrieved from a remote system, the EBCDIC characters are automatically converted to ASCII.
fileImage	The file is transferred without any modification. This is the default file transfer type, and should be used when transferring binary (non-text) data.
fileText	The same value as **fileAscii**.
fileBinary	The same value as **fileImage**.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace

FileTransfer.FtpChannelMode Enumeration

Specifies the channel mode specified by setting the ChannelMode property.

[Visual Basic]

```
Public Enum FileTransfer.FtpChannelMode
```

[C#]

```
public enum FileTransfer.FtpChannelMode
```

Members

Member Name	Description
channelClear	Data sent and received on this channel should not be encrypted.
channelSecure	Data sent and received on this channel should be encrypted. Specifying this option requires that a secure connection has already been established with the server.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace

FileTransfer.FtpChannelType Enumeration

Specifies the channel used by the ChannelMode property.

[Visual Basic]
```
Public Enum FileTransfer.FtpChannelType
```

[C#]
```
public enum FileTransfer.FtpChannelType
```

Members

Member Name	Description
channelCommand	The communication channel used to send commands to the server and receive command result and status information from the server.
channelData	The communication channel used to send or receive data during a file transfer.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace

FileTransfer.FtpDirectoryFormat Enumeration

A value which specifies the format of a directory listing returned by the server.

[Visual Basic]
```
Public Enum FileTransfer.FtpDirectoryFormat
```

[C#]
```
public enum FileTransfer.FtpDirectoryFormat
```

Remarks

Values other than **formatAuto** should only be set if the class library cannot automatically determine the directory format returned by the server. The default directory format is determined both by the server's operating system and by analyzing the format of the data returned by the server. If the class is unable to automatically determine the format, it will attempt to parse the list of files as though it is a UNIX style listing.

Members

Member Name	Description
formatAuto	This value specifies that the control should automatically determine the format of the file lists returned by the server. It is recommended that most applications use this value and allow the control to automatically determine the appropriate file listing format used by the server.
formatUnix	This value specifies that the server returns file lists in the format commonly used by UNIX servers. Note that many servers can be configured to return file listings in this format, even if they are not actually a UNIX based platform. Consult the technical reference documentation for your server for more information.
formatMsdos	This value specifies that the server returns file lists in the format commonly used by MS-DOS based systems. This includes Windows NT servers. Long file names will be returned if supported by the underlying filesystem, such as NTFS or FAT32.
formatVms	This value specifies that the server returns file lists in the format commonly used by VMS servers. Note that VMS servers can be configured to return a standard UNIX style listing in additional to the default VMS format.
formatSterling1	This value specifies that the server returns file listings in a proprietary format used by the Sterling server, which is used for EDI (Electronic Data Interchange) applications. This format uses a 13 byte status code.
formatSterling2	This value specifies that the server returns file

	listings in a proprietary format used by the Sterling server, which is used for EDI (Electronic Data Interchange) applications. This format uses a 10 byte status code.
formatNetware	This value specifies that the server returns file listings in a proprietary format used by NetWare servers. The format is similar to UNIX style listings except that file access and permissions are indicated by letter codes enclosed in brackets. This is the default format selected if the server identifies itself as a NetWare system.
formatMlsd	This value specifies that the server should return file listings in a machine-independent format as defined by RFC 3659. This format specifies file information as a sequence of name and value pairs, with the same format being used regardless of the operating system that the server is hosted on. Note that not all servers support this format, and some proxy servers may reject the command even if the remote server supports its use.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace

FileTransfer.FtpFeatures Enumeration

Specifies the server features that are available for the current client session.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum FileTransfer.FtpFeatures
```

[C#]
```
[Flags]
public enum FileTransfer.FtpFeatures
```

Remarks

When a client connection is first established, all features are enabled by default. However, as the client issues commands to the server, if the server reports that the command is unrecognized that feature will automatically be disabled in the client.

For example, the first time an application calls the **GetFileSize** method to determine the size of a file, the class library will try to use the SIZE command. If the server reports that the SIZE command is not available, that feature will be disabled and the class will not use the command again during the session unless it is explicitly re-enabled. This is designed to prevent the class from repeatedly sending invalid commands to a server, which may result in the server aborting the connection.

Members

Member Name	Description	Value
featureSIZE	The server supports the SIZE command to determine the size of a file. If this feature is not enabled, the library will attempt to use the STAT command to determine the file size.	1
featureSTAT	The server supports using the STAT command to return information about a specific file. If this feature is not enabled, the client may not be able to obtain information about a specific file such as its size, permissions or modification time.	2
featureMDTM	The server supports the MDTM command to obtain information about the modification time for a specific file. This command may also be used to set the file time on the server.	4
featureREST	The server supports restarting file transfers using the REST command. If this feature is not enabled, the client will not be able to restart file transfers and must upload or download the complete file.	8

featureSITE	The server supports site specific commands using the SITE command. If this feature is not enabled, no site specific commands will be sent to the server.	16
featureIDLE	The server supports setting the idle timeout period using the SITE IDLE command to specify the number of seconds that the client may idle before the server terminates the connection.	32
featureCHMOD	The server supports modifying the permissions of a specific file using the SITE CHMOD command. If this feature is not enabled, the client will not be able to set the permissions for a file.	64
featureAUTH	The server supports explicit SSL sessions using the AUTH command. If this feature is not enabled, the client will only be able to connect to a secure server that uses implicit SSL connections. Changing this feature has no effect on standard, non-secure connections.	128
featurePBSZ	The server supports the PBSZ command which specifies the buffer size used with secure data connections. If this feature is disabled, it may prevent the client from changing the protection level on the data channel. Changing this feature has no effect on standard, non-secure connections.	256
featurePROT	The server supports the PROT command which specifies the protection level for the data channel. If this feature is disabled, the client will be unable to change the protection level on the data channel. Changing this feature has no effect on standard, non-secure connections.	512
featureCCC	The server supports the CCC command which returns the command channel to a non-secure mode. Changing this feature has no effect on standard, non-secure connections.	1024
featureHOST	The server supports the HOST command which enables a client to specify the hostname after establishing a connection with a server that supports virtual hosting.	2048

featureMLST	The server supports the MLST command which returns status information for files. If this feature is enabled, the MLST command will be used instead of the STAT command.	4096
featureMFMT	The server supports the MFMT command which is used to change the last modification time for a file. If this command is supported, it is used instead of the MDTM command to change the modification time for a file.	8192
featureXCRC	The server supports the XCRC command which returns the CRC32 checksum for the contents of a specified file. This command is used for file verification.	16384
featureXMD5	The server supports the XMD5 command which returns an MD5 hash for the contents of a specified file. This command is used for file verification.	32768
featureLANG	The server supports the LANG command which sets the language used for the current client session. Command responses and file naming conventions will use the specified language.	65536
featureUTF8	The server supports the OPTS UTF-8 command which specifies UTF-8 encoding when specifying filenames. This feature is typically used in conjunction with setting the default language for the client session.	131072
featureXQUOTA	The server supports the XQUOTA command which returns quota information for the current client session.	262144
featureUTIME	The server supports the UTIME command which is used to change the last modification time for a specified file.	524288

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace

FileTransfer.HttpVersion Enumeration

Specifies the protocol versions supported by the FileTransfer class.

[Visual Basic]

```
Public Enum FileTransfer.HttpVersion
```

[C#]

```
public enum FileTransfer.HttpVersion
```

Remarks

The FileTransfer class uses the **HttpVersion** enumeration to specify the protocol version to be used when establishing a connection to the server.

Members

Member Name	Description
version09	Version 0.9 of the protocol. This value specifies that the client should use the preliminary protocol version which only supports the use of the GET command. Header fields are not supported with this version of the protocol.
version10	Version 1.0 of the protocol.
version11	Version 1.1 of the protocol.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace

FileTransfer.SecureCipherAlgorithm Enumeration

Specifies the encryption algorithms that the FileTransfer class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum FileTransfer.SecureCipherAlgorithm
```

[C#]

```
[Flags]
public enum FileTransfer.SecureCipherAlgorithm
```

Remarks

The FileTransfer class uses the **SecureCipherAlgorithm** enumeration to identify which encryption algorithm was selected when a secure connection was established with the server.

Members

Member Name	Description	Value
cipherNone	No cipher has been selected. A secure connection has not been established with the server.	0
cipherRC2	The RC2 block cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.	1
cipherRC4	The RC4 stream cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.	2
cipherRC5	The RC5 block cipher was selected. This is a variable key length cipher which supports keys up to 2040 bits, in 8-bit increments.	4
cipherDES	The DES (Data Encryption Standard) block cipher was selected. This is a fixed key length cipher using 56-bit keys.	8
cipherDES3	The Triple DES block cipher was selected. This cipher encrypts the data three times using different keys, effectively using a 168-bit key length.	16
cipherDESX	A variant of the DES block cipher which XORs an extra 64-bits of the key before and after the plaintext has been encrypted, increasing the key size to 184 bits.	32
cipherAES	The Advanced Encryption Standard	64

	cipher (also known as the Rijndael cipher) is a fixed block size cipher which use a key size of 128, 192 or 256 bits. This cipher is supported on Windows XP SP3 SP3 and later versions of the operating system.	
cipherSkipjack	The Skipjack block cipher was selected. This is a fixed key length cipher, using 80-bit keys.	128
cipherBlowfish	The Blowfish block cipher was selected. This is a variable key length cipher up to 448 bits, using a 64-bit block size.	256

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace

FileTransfer.SecureHashAlgorithm Enumeration

Specifies the hash algorithms that the FileTransfer class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum FileTransfer.SecureHashAlgorithm
```

[C#]

```
[Flags]
public enum FileTransfer.SecureHashAlgorithm
```

Remarks

The FileTransfer class uses the **SecureHashAlgorithm** enumeration to identify the message digest (hash) algorithm that was selected when a secure connection was established with the server.

Members

Member Name	Description	Value
hashNone	No hash algorithm has been selected. This is not a secure connection with the server.	0
hashMD5	The MD5 algorithm was selected. This algorithm produces a 128-bit message digest. This algorithm is no longer considered to be cryptographically secure.	1
hashSHA	The SHA-1 algorithm was selected. This algorithm produces a 160-bit message digest. This algorithm is no longer considered to be cryptographically secure.	2
hashSHA256	The SHA-256 algorithm was selected. This algorithm produces a 256-bit message digest.	4
hashSHA384	The SHA-384 algorithm was selected. This algorithm produces a 384-bit message digest.	8
hashSHA512	The SHA-512 algorithm was selected. This algorithm produces a 512-bit message digest.	16

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace

FileTransfer.SecureKeyAlgorithm Enumeration

Specifies the key exchange algorithms that the FileTransfer class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum FileTransfer.SecureKeyAlgorithm
```

```
[C#]
[Flags]
public enum FileTransfer.SecureKeyAlgorithm
```

Remarks

The FileTransfer class uses the **SecureKeyAlgorithm** enumeration to identify the key exchange algorithm that was selected when a secure connection was established with the server.

Members

Member Name	Description	Value
keyExchangeNone	No key exchange algorithm has been selected. This is not a secure connection with the server.	0
keyExchangeRSA	The RSA public key exchange algorithm has been selected.	1
keyExchangeKEA	The KEA public key exchange algorithm has been selected. This is an improved version of the Diffie-Hellman public key algorithm.	2
keyExchangeDH	The Diffie-Hellman public key exchange algorithm has been selected.	4
keyExchangeECDH	The Elliptic Curve Diffie-Hellman key exchange algorithm was selected. This is a variant of the Diffie-Hellman algorithm which uses elliptic curve cryptography. This key exchange algorithm is only supported on Windows XP SP3 SP3 and later versions of the operating system.	8

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace

FileTransfer.SecurityCertificate Enumeration

Specifies the security certificate status values that may be returned by the FileTransfer class.

[Visual Basic]

```
Public Enum FileTransfer.SecurityCertificate
```

[C#]

```
public enum FileTransfer.SecurityCertificate
```

Remarks

The FileTransfer class uses the **SecurityCertificate** enumeration to identify the current status of the certificate that was provided by the server when a secure connection was established.

Members

Member Name	Description
certificateNone	No certificate information is available. A secure connection was not established with the server.
certificateValid	The certificate is valid.
certificateNoMatch	The certificate is valid, however the domain name specified in the certificate does not match the domain name of the server. The application can examine the **CertificateSubject** property to determine the site the certificate was issued to.
certificateExpired	The certificate has expired and is no longer valid. The application can examine the **CertificateExpires** property to determine when the certificate expired.
certificateRevoked	The certificate has been revoked and is no longer valid. It is recommended that the application immediately terminate the connection if this status is returned.
certificateUntrusted	The certificate has not been issued by a trusted authority, or the certificate is not trusted on the local host. It is recommended that the application immediately terminate the connection if this status is returned.
certificateInvalid	The certificate is invalid. This typically indicates that the internal structure of the certificate is damaged. It is recommended that the application immediately terminate the connection if this status is returned.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace

FileTransfer.SecurityProtocols Enumeration

Specifies the security protocols that the FileTransfer class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum FileTransfer.SecurityProtocols
```

```
[C#]
[Flags]
public enum FileTransfer.SecurityProtocols
```

Remarks

The FileTransfer class uses the **SecurityProtocols** enumeration to specify one or more security protocols to be used when establishing a connection with a server. Multiple protocols may be specified if necessary and the actual protocol used will be negotiated with the server. It is recommended that most applications use **protocolDefault** when creating a secure connection.

Members

Member Name	Description	Value
protocolNone	No security protocol will be used, a secure connection will not be established.	0
protocolSSL2	The SSL 2.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections.	1
protocolSSL3	The SSL 3.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections. In most cases, this protocol is only selected if TLS is not supported by the server.	2
protocolTLS10	The TLS 1.0 protocol has been selected. This version of the protocol is commonly used by older servers and is the only version of TLS supported on Windows platforms prior to Windows 7 SP1 and Windows Server 2008 R2.	4
protocolTLS11	The TLS 1.1 protocol has been selected. This version of TLS is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of the	8

	operating system.	
protocolTLS12	The TLS 1.2 protocol has been selected. This is the default version of the protocol and is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of Windows. It is recommended that you use this version of TLS.	16
protocolSSH1	The Secure Shell 1.0 protocol has been selected. This version of the protocol has been deprecated and is no longer widely used. It is not recommended that this version of the protocol be used to establish a connection.	256
protocolSSH2	The Secure Shell 2.0 protocol has been selected. This is the most commonly used version of the protocol. It is recommended that this version of the protocol be used unless the server explicitly requires the client to use an earlier version.	512
protocolSSL	Any version of the Secure Sockets Layer (SSL) protocol should be used. The actual protocol version used will be negotiated with the remote host.	3
protocolTLS	Any version of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with with preference for TLS 1.2.	28
protocolTLS1	Version 1.0, 1.1 or 1.2 of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with preference for TLS 1.2.	28
protocolSSH	Any version of the the Secure Shell (SSH) protocol should be used. The actual protocol version used will be negotiated with the remote host.	768
protocolDefault	The default selection of security protocols will be used when establishing a connection. The TLS 1.2, 1.1 and 1.0 protocols will be negotiated with the host, in that order of preference. This option will always request the latest version of the preferred security protocols and is the recommended	16

	value.	
protocolUnknown	An unknown or unsupported security protocol has been specified. This value indicates an error condition.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace

FileTransfer.TraceOptions Enumeration

Specifies the logging options that the FileTransfer class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum FileTransfer.TraceOptions
```

```
[C#]
[Flags]
public enum FileTransfer.TraceOptions
```

Remarks

The FileTransfer class uses the **TraceOptions** enumeration to specify what kind of debugging information is written to the trace logfile. These options are only meaningful when trace logging is enabled by setting the **Trace** property to **true**.

Members

Member Name	Description	Value
traceDefault	The default trace logging option. This is the same as specifying the **traceInfo** option.	0
traceInfo	All network function calls are written to the trace file. This is the default value.	0
traceError	Only those network function calls which fail are recorded in the trace file.	1
traceWarning	Only those network function calls which fail, or return values which indicate a warning, are recorded in the trace file.	2
traceHexDump	All network function calls are written to the trace file, plus all the data that is sent or received is displayed, in both ASCII and hexadecimal format.	4
traceProcess	All function calls in the current process are logged, rather than only those functions in the current thread. This option is useful for multithreaded applications that are using worker threads.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace

FileTransfer.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see FileTransfer.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.FileTransfer.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class FileTransfer.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class FileTransfer.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the FileTransfer class.

Example

```
<Assembly: SocketTools.FileTransfer.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.FileTransfer.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

FileTransfer.RuntimeLicenseAttribute Members | SocketTools Namespace

FileTransfer.RuntimeLicenseAttribute Members

FileTransfer.RuntimeLicenseAttribute overview

Public Instance Constructors

FileTransfer.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FileTransfer.RuntimeLicenseAttribute Class | SocketTools Namespace

FileTransfer.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public FileTransfer.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the FileTransfer class.

See Also

FileTransfer.RuntimeLicenseAttribute Class | SocketTools Namespace

FileTransfer.RuntimeLicenseAttribute Properties

The properties of the **FileTransfer.RuntimeLicenseAttribute** class are listed below. For a complete list of **FileTransfer.RuntimeLicenseAttribute** class members, see the FileTransfer.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

FileTransfer.RuntimeLicenseAttribute Class | SocketTools Namespace

FileTransfer.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

FileTransfer.RuntimeLicenseAttribute Class | SocketTools Namespace

FileTransfer.ThreadModelAttribute Class

Attribute that defines the threading model for the class.

For a list of all members of this type, see FileTransfer.ThreadModelAttribute Members.

System.Object
  System.Attribute
    SocketTools.FileTransfer.ThreadModelAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False, Inherited:=True)>
Public Class FileTransfer.ThreadModelAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False, Inherited=True)]
public class FileTransfer.ThreadModelAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The ThreadModel attribute is used to define the threading model that is to be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#.

Example

```
<Assembly:
SocketTools.FileTransfer.ThreadModel(SocketTools.FileTransfer.ThreadModelAttribute.Model.SingleThread)>
```

```
[assembly:
SocketTools.FileTransfer.ThreadModel(SocketTools.FileTransfer.ThreadModelAttribute.Model.SingleThread)]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

FileTransfer.ThreadModelAttribute Members | SocketTools Namespace | ThreadModel Property (SocketTools.FileTransfer)

FileTransfer.ThreadModelAttribute Members

FileTransfer.ThreadModelAttribute overview

Public Instance Constructors

FileTransfer.ThreadModelAttribute Constructor	Constructor for the ThreadModel attribute which defines the threading model.

Public Instance Properties

ThreadModel	Returns the threading model used by the class.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FileTransfer.ThreadModelAttribute Class | SocketTools Namespace | ThreadModel Property (SocketTools.FileTransfer)

FileTransfer.ThreadModelAttribute Constructor

Constructor for the ThreadModel attribute which defines the threading model.

[Visual Basic]

```
Public Sub New( _
   ByVal threadModel As Model _
)
```

[C#]

```
public FileTransfer.ThreadModelAttribute(
   Model threadModel
);
```

Parameters

threadModel

A Model enumeration value which specifies the threading model which will be used when creating an instance of the class. A value of zero specifies a single threaded model, while a non-zero value specifies a free threaded model.

Remarks

The **ThreadModel** attribute specifies the threading model that is used by the class instance when a connection is established. The default threading model is single threaded, which specifies that only the thread that established the connection should be permitted to invoke methods.

It is important to note that the single threading model does not limit the application to a single thread of execution. When a session is established using the **Connect** method, that session is attached to the thread that created it. From that point on, until the session is terminated, only the owner may invoke methods in that instance of the class. The ownership of the class instance may be transferred from one thread to another using the **AttachThread** method.

Setting this attribute to a non-zero value disables certain internal safety checks that are performed by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads. The application must ensure that no two threads will attempt to invoke a blocking method at the same time. In other words, if one thread invokes a method, the application must ensure that another thread will not attempt to invoke any other method at the same time using the same instance of the class.

Changing the value of this attribute will also change the default value for the **ThreadModel** property for all instances of the class.

See Also

FileTransfer.ThreadModelAttribute Class | SocketTools Namespace

FileTransfer.ThreadModelAttribute Properties

The properties of the **FileTransfer.ThreadModelAttribute** class are listed below. For a complete list of **FileTransfer.ThreadModelAttribute** class members, see the FileTransfer.ThreadModelAttribute Members topic.

Public Instance Properties

ThreadModel	Returns the threading model used by the class.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

FileTransfer.ThreadModelAttribute Class | SocketTools Namespace | ThreadModel Property (SocketTools.FileTransfer)

FileTransfer.ThreadModelAttribute.ThreadModel Property

Returns the threading model used by the class.

[Visual Basic]

```
Public Property ThreadModel As Model
```

[C#]

```
public FileTransfer.ThreadModelAttribute.Model ThreadModel {get; set;}
```

Property Value

A Model enumeration value which specifies the threading model which will be used when an instance of the class is created.

See Also

FileTransfer.ThreadModelAttribute Class | SocketTools Namespace

FileTransfer.ThreadModelAttribute.Model Enumeration

Specifies the threading model used by the class instance.

[Visual Basic]
```
Public Enum FileTransfer.ThreadModelAttribute.Model
```

[C#]
```
public enum FileTransfer.ThreadModelAttribute.Model
```

Remarks

The threading model **SingleThread** does not limit the application to a single thread of execution. It specifies that only a single thread may invoke methods in a class instance. When a session is established using the **Connect** method, that session is attached to the thread that created it. From that point on, until the session is terminated, only the owner may invoke methods in that instance of the class. The ownership of the class instance may be transferred from one thread to another using the **AttachThread** method.

The threading model **FreeThread** disables certain internal safety checks that are performed by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads. The application must ensure that no two threads will attempt to invoke a blocking method at the same time. In other words, if one thread invokes a method, the application must ensure that another thread will not attempt to invoke any other method at the same time using the same instance of the class.

Members

Member Name	Description
SingleThread	Methods in the class instance may only be invoked by a single thread. This threading model specifies that only the thread which established the connection should be permitted to invoke methods. This is the default threading model.
FreeThread	Methods in the class instance may be invoked by any thread. This threading model permits methods to be invoked across multiple threads without being explicitly attached to the object.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

SocketTools Namespace

FileTransferException Class

The exception that is thrown when a client error occurs.

For a list of all members of this type, see FileTransferException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.FileTransferException

[Visual Basic]

```
Public Class FileTransferException
    Inherits ApplicationException
```

[C#]

```
public class FileTransferException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A FileTransferException is thrown by the FileTransfer class when an error occurs.

The default constructor for the FileTransferException class sets the **ErrorCode** property to the last error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FileTransfer (in SocketTools.FileTransfer.dll)

See Also

FileTransferException Members | SocketTools Namespace

FileTransferException Members

FileTransferException overview

Public Instance Constructors

FileTransferException	Overloaded. Initializes a new instance of the FileTransferException class.

Public Instance Properties

HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FileTransferException Class | SocketTools Namespace

FileTransferException Constructor

Initializes a new instance of the FileTransferException class with the last client error code.

Overload List

Initializes a new instance of the FileTransferException class with the last client error code.

public FileTransferException();

Initializes a new instance of the FileTransferException class with a specified error number.

public FileTransferException(int);

Initializes a new instance of the FileTransferException class with a specified error message.

public FileTransferException(string);

Initializes a new instance of the FileTransferException class with a specified error message and a reference to the inner exception that is the cause of this exception.

public FileTransferException(string,Exception);

See Also

FileTransferException Class | SocketTools Namespace

FileTransferException Constructor ()

Initializes a new instance of the FileTransferException class with the last client error code.

[Visual Basic]
```
Overloads Public Sub New()
```

[C#]
```
public FileTransferException();
```

Remarks

The ctor constructor sets the **ErrorCode** property to the last client error that occurred. For more information about the errors that may occur, refer to the FileTransfer.ErrorCode enumeration.

See Also

FileTransferException Class | SocketTools Namespace | FileTransferException Constructor Overload List

FileTransferException Constructor (String)

Initializes a new instance of the FileTransferException class with a specified error message.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]

```
public FileTransferException(
   string message
);
```

Parameters

message
The error message that explains the reason for the exception.

Remarks

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

FileTransferException Class | SocketTools Namespace | FileTransferException Constructor Overload List

FileTransferException Constructor (String, Exception)

Initializes a new instance of the FileTransferException class with a specified error message and a reference to the inner exception that is the cause of this exception.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal inner As Exception _
)
```

[C#]

```
public FileTransferException(
   string message,
   Exception inner
);
```

Parameters

message

The error message that explains the reason for the exception.

inner

The exception that is the cause of the current exception. If the ***innerException*** parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

FileTransferException Class | SocketTools Namespace | FileTransferException Constructor Overload List

FileTransferException Constructor (Int32)

Initializes a new instance of the FileTransferException class with a specified error number.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

[C#]

```
public FileTransferException(
   int code
);
```

Parameters

code

An integer value which specifies an error code.

Remarks

This constructor sets the **ErrorCode** property to the specified error code. For more information about the errors that may occur, refer to the FileTransfer.ErrorCode enumeration.

See Also

FileTransferException Class | SocketTools Namespace | FileTransferException Constructor Overload List

FileTransferException Properties

The properties of the **FileTransferException** class are listed below. For a complete list of **FileTransferException** class members, see the FileTransferException Members topic.

Public Instance Properties

HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

FileTransferException Class | SocketTools Namespace

FileTransferException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]

```
Overrides Public ReadOnly Property Message As String
```

[C#]

```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

FileTransferException Class | SocketTools Namespace

FileTransferException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]
```
Public ReadOnly Property Number As Integer
```

[C#]
```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception. If a network error occurs, this value is the same as the values returned by the Windows Sockets API. For more information about network error codes, see the Windows Socket Version 2 API error code documentation in MSDN.

See Also

FileTransferException Class | SocketTools Namespace

FileTransferException Methods

The methods of the **FileTransferException** class are listed below. For a complete list of **FileTransferException** class members, see the FileTransferException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FileTransferException Class | SocketTools Namespace

FileTransferException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]
```
Overrides Public Function ToString() As String
```

[C#]
```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

FileTransferException Class | SocketTools Namespace

FtpClient Class

Implements the File Transfer Protocol.

For a list of all members of this type, see FtpClient Members.

System.Object
  SocketTools.FtpClient

```
[Visual Basic]
Public Class FtpClient
    Implements IDisposable
```

```
[C#]
public class FtpClient : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The FtpClient class provides a comprehensive interface to the File Transfer Protocol which supports both high level operations, such as uploading or downloading files, as well as a collection of lower-level file I/O functions. In addition to file transfers, an application can create, rename and delete files and directories, search for files using wildcards and perform other common file management functions. The class library has three distinct groups of functionality:

File Transfer
Methods which enable an application to upload and download files, as well as send and receive file data using memory buffers. This gives your program the flexibility of handling the data either on disk or in memory, depending on the best needs of your application. If your program needs to transfer more than one file at a time, there are also methods which will automatically download or upload multiple files in a single method call.

File Management
In addition to transferring files, the class can be used to manage files on the server. Methods are provided to delete, rename and move files between directories. For servers that support specific protocol extensions, advanced features such as getting or setting a remote file's modification time or access permissions are also supported. If a server supports site-specific commands, such as the ability to submit a file as job on the server, the control supports this by enabling you to send custom commands to the server and then process the information that it returns.

Directory Management
The class can be used to manage directories as well as files on the server. The application can open a directory and return a list of the files that it contains, as well as create new directories and delete empty ones. The class understands a number of different directory listing formats, including those typically used on UNIX and Linux based systems, Windows server platforms, NetWare servers and VMS systems.

This class supports secure file transfers using TLS 1.2 and the SSH 2.0 protocol using SFTP.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

FtpClient Members | SocketTools Namespace

FtpClient Members

FtpClient overview

Public Static (Shared) Fields

ftpPortDefault	A constant value which specifies the default port number.
ftpPortSecure	A constant value which specifies the default port number for a secure connection using the SSL or TLS protocols.
ftpPortSSH	A constant value which specifies the default port number for a secure connection using the SSH1 or SSH2 protocols.
ftpTimeout	A constant value which specifies the default timeout period.

Public Static (Shared) Methods

ErrorText	Returns the description of an error code.

Public Instance Constructors

FtpClient Constructor	Initializes a new instance of the FtpClient class.

Public Instance Properties

Account	Get or sets a value that specifies the account name for the current user.
ActivePorts	Gets and sets the port numbers used for active mode file transfers.
AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
Blocking	Gets and sets a value which indicates if the client is in blocking mode.
BufferSize	Gets and sets the size of the internal send and receive buffer that will be used during data transfers.
CertificateExpires	Get a value that specifies the date that the security certificate expires.
CertificateIssued	Get a value that specifies the date that the security certificate was issued.
CertificateIssuer	Get a value that provides information about the organization that issued the certificate.
CertificateName	Gets and sets a value that specifies the name of the client certificate.
CertificatePassword	Gets and sets the password associated with the client certificate.

CertificateStatus	Gets a value which indicates the status of the security certificate returned by the remote host.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateSubject	Gets a value that provides information about the organization that the server certificate was issued to.
CertificateUser	Gets and sets the user that owns the client certificate.
ChannelMode	Set or return the security mode for the specified communications channel.
CipherStrength	Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.
DirectoryFormat	Gets and sets a value which specifies the current directory format type.
Encoding	Gets and sets the character encoding that is used when a file name is sent to the server.
Features	Gets and sets the features that are currently enabled for the current session.
FileMask	Gets and sets the value which specifies the default wildcard file mask.
FileType	Gets and sets a value which specifies the type of file that is being transferred.
Fingerprint	Gets a value that can be used to uniquely identify the server.
Handle	Gets a value that specifies the client handle allocated for the current session.
HashStrength	Gets a value which specifies the length of the message digest that was selected for a secure connection.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsConnected	Gets a value which indicates if a connection to the remote host has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsReadable	Gets a value which indicates if there is data available to be read from the socket connection to

	the server.
IsWritable	Gets a value which indicates if data can be written to the client without blocking.
KeepAlive	Gets and sets a Boolean value which specifies if the client should attempt to keep the server connection alive.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets the local Internet address that the client is bound to.
Localize	Gets and sets a value which specifies if time and dates should be adjusted for the current timezone.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
Options	Gets and sets a value which specifies one or more client options.
ParseList	Gets and sets a value that specifies if directory listings should be automatically parsed
Passive	Gets and sets a value which specifies if passive mode file transfers should be enabled.
Password	Gets and sets the password used to authenticate the client session.
Priority	Gets and sets a value which specifies the priority of file transfers.
ProxyHost	Gets and sets the hostname or IP address of a proxy server.
ProxyPassword	Gets and sets the password used to authenticate the connection to a proxy server.
ProxyPort	Gets and sets a value that specifies the proxy server port number.
ProxyType	Gets and sets the type of proxy server the client will use to establish a connection.
ProxyUser	Gets and sets the username used to authenticate the connection to a proxy server.
RemoteFile	Gets and sets the file name specified in the current URL.
RemotePath	Gets and sets the path specified in the current URL.
RemotePort	Gets and sets a value which specifies the remote

	port number.
RemoteService	Gets and sets a value which specifies the remote service.
ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.
Secure	Gets and sets a value which specifies if a secure connection is established.
SecureCipher	Gets a value that specifies the encryption algorithm used for a secure connection.
SecureHash	Gets a value that specifies the message digest algorithm used for a secure connection.
SecureKeyExchange	Gets a value that specifies the key exchange algorithm used for a secure connection.
SecureProtocol	Gets and sets a value which specifies the protocol used for a secure connection.
Status	Gets a value which specifies the current status of the client.
System	Gets a string value which identifies the server.
TaskCount	Get the number of active background file transfers.
TaskId	Get the task identifier for the last background file transfer.
TaskList	Get an array of active background task identifiers.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
TimeZone	Gets and sets the current timezone offset in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
TransferBytes	Gets a value which specifies the number of bytes transferred to or from the remote server.
TransferRate	Gets a value which specifies the data transfer rate

	in bytes per second.
TransferTime	Gets a value which specifies the number of seconds elapsed during a data transfer.
URL	Gets and sets the current URL used to access a file on the server.
UserName	Gets and sets the username used to authenticate the client session.
Version	Gets a value which returns the current version of the FtpClient class library.

Public Instance Methods

AddFileType	Associate a file name extension with a specific file type.
AsyncGetFile	Overloaded. Download a file from the server to the local system in the background.
AsyncPutFile	Overloaded. Upload a file from the local system to the server in the background.
AttachThread	Attach an instance of the class to the current thread
Cancel	Cancel the current blocking client operation.
ChangeDirectory	Change the current working directory on the remote server.
CloseDirectory	Close the directory that was previously opened with the OpenDirectory method.
CloseFile	Close the file that was previously opened with the OpenFile method.
Command	Overloaded. Send a custom command to the server.
Connect	Overloaded. Establish a connection with a remote host.
CreateFile	Create a new file or overwrite an existing file.
DeleteFile	Delete a file on the remote server.
Disconnect	Terminate the connection with the remote server.
Dispose	Overloaded. Releases all resources used by FtpClient.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
FileList	Overloaded. Return a list of files on the remote host.
GetData	Overloaded. Transfers the contents of a file on the server and stores it in a byte array.
GetDirectory	Return the current working directory.

GetFile	Overloaded. Download a file from the server to the local system.
GetFileList	Overloaded. Returns an unparsed list of files in the specified directory.
GetFilePermissions	Return the access permissions for a file on the remote system.
GetFileSize	Overloaded. Returns the size of the specified file on the remote server.
GetFileStatus	Returns status information about the specified file.
GetFileTime	Overloaded. Returns the modification date and time for specified file on the remote server.
GetFirstFile	Overloaded. Get information about the first file in a directory listing returned by the server.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetMultipleFiles	Overloaded. Download multiple files from the server to the local system using a wildcard mask.
GetNextFile	Overloaded. Get information about the next file in a directory listing returned by the server.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the FtpClient class.
Login	Overloaded. Login to the remote server.
Logout	Log the current user off the server.
MakeDirectory	Create a new directory on the server.
OpenDirectory	Overloaded. Open the specified directory on the server.
OpenFile	Overloaded. Open an existing file or create a new file on the server.
PutData	Overloaded. Transfers data from a byte array and stores it in a file on the remote server.
PutFile	Overloaded. Upload a file from the local system to the server.
PutMultipleFiles	Overloaded. Upload multiple files from the local system to the server using a wildcard mask.
Read	Overloaded. Read file data from the server and store it in a byte array.
RemoveDirectory	Remove a directory on the server.
RenameFile	Change the name of a file on the server.
Reset	Reset the internal state of the object, resetting all

	properties to their default values.
SetFilePermissions	Change the access permissions for a file on the server.
SetFileTime	Changes the modification date and time for a file on the server.
TaskAbort	Overloaded. Abort the specified asynchronous task.
TaskDone	Overloaded. Determine if an asynchronous task has completed.
TaskResume	Overloaded. Resume execution of an asynchronous task.
TaskSuspend	Overloaded. Suspend execution of an asynchronous task.
TaskWait	Overloaded. Wait for an asynchronous task to complete.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
VerifyFile	Overloaded. Verify that the contents of a file on the local system are the same as the specified file on the server.
Write	Overloaded. Write one or more bytes of data to the server.

Public Instance Events

OnCancel	Occurs when a blocking client operation is canceled.
OnCommand	Occurs when the client sends a command to the remote host and receives a reply indicating the result of that command.
OnConnect	Occurs when a connection is established with the remote host.
OnDisconnect	Occurs when the remote host disconnects from the local system.
OnError	Occurs when an socket operation fails.
OnFileList	Occurs when a directory listing is parsed by the class.
OnGetFile	Occurs when a file download has been initiated.
OnLastFile	Occurs when the last file in a directory listing has been processed.
OnProgress	Occurs as a data stream is being read or written to the socket.

OnPutFile	Occurs when a file upload is initiated.
OnRead	Occurs when data is available to be read from the socket.
OnTaskBegin	Occurs when an asynchronous task begins execution.
OnTaskEnd	Occurs when an asynchronous task completes.
OnTaskRun	Occurs while a background task is active.
OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
OnWrite	Occurs when data can be written to the socket.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the FtpClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpClient Class | SocketTools Namespace

FtpClient Constructor

Initializes a new instance of the FtpClient class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FtpClient();
```

See Also

FtpClient Class | SocketTools Namespace

FtpClient.FileInformation Structure

This structure is used by the GetFirstFile and GetNextFile methods to return information about a file on the server.

For a list of all members of this type, see FtpClient.FileInformation Members.

System.Object
  System.ValueType
    SocketTools.FtpClient.FileInformation

[Visual Basic]
```
Public Structure FtpClient.FileInformation
```

[C#]
```
public struct FtpClient.FileInformation
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

FtpClient.FileInformation Members | SocketTools Namespace

FtpClient.FileInformation Members

FtpClient.FileInformation overview

Public Instance Fields

FileDate	Specifies the date and time the file was created or last modified.
FileGroup	Specifies the name of the group that owns the file.
FileLinks	Specifies the number of links to the file.
FileName	Specifies the name of the file.
FileOwner	Specifies the name of the file owner.
FilePerms	Specifies the permissions for the file.
FileSize	Specifies the size of the file in bytes.
FileVersion	Specifies the number of revisions made to the file.
IsDirectory	Specifies if the file is a directory or regular file.

Public Instance Methods

Equals (inherited from ValueType)	Indicates whether this instance and a specified object are equal.
GetHashCode (inherited from ValueType)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from ValueType)	Returns the fully qualified type name of this instance.

See Also

FtpClient.FileInformation Class | SocketTools Namespace

FtpClient.FileInformation Fields

The fields of the **FtpClient.FileInformation** structure are listed below. For a complete list of **FtpClient.FileInformation** structure members, see the FtpClient.FileInformation Members topic.

Public Instance Fields

FileDate	Specifies the date and time the file was created or last modified.
FileGroup	Specifies the name of the group that owns the file.
FileLinks	Specifies the number of links to the file.
FileName	Specifies the name of the file.
FileOwner	Specifies the name of the file owner.
FilePerms	Specifies the permissions for the file.
FileSize	Specifies the size of the file in bytes.
FileVersion	Specifies the number of revisions made to the file.
IsDirectory	Specifies if the file is a directory or regular file.

See Also

FtpClient.FileInformation Class | SocketTools Namespace

FtpClient.FileInformation.FileDate Field

Specifies the date and time the file was created or last modified.

[Visual Basic]

```
Public FileDate As Date
```

[C#]

```
public DateTime FileDate;
```

Remarks

Some file servers may not return a specific time value if the file was last created or modified more than six months ago. In that case, only the date will be returned.

See Also

FtpClient.FileInformation Class | SocketTools Namespace

FtpClient.FileInformation.FileGroup Field

Specifies the name of the group that owns the file.

[Visual Basic]

```
Public FileGroup As String
```

[C#]

```
public string FileGroup;
```

See Also

FtpClient.FileInformation Class | SocketTools Namespace

FtpClient.FileInformation.FileLinks Field

Specifies the number of links to the file.

[Visual Basic]

```
Public FileLinks As Integer
```

[C#]

```
public int FileLinks;
```

See Also

FtpClient.FileInformation Class | SocketTools Namespace

FtpClient.FileInformation.FileName Field

Specifies the name of the file.

[Visual Basic]

```
Public FileName As String
```

[C#]

```
public string FileName;
```

See Also

FtpClient.FileInformation Class | SocketTools Namespace

FtpClient.FileInformation.FileOwner Field

Specifies the name of the file owner.

[Visual Basic]

```
Public FileOwner As String
```

[C#]

```
public string FileOwner;
```

See Also

FtpClient.FileInformation Class | SocketTools Namespace

FtpClient.FileInformation.FilePerms Field

Specifies the permissions for the file.

[Visual Basic]

```
Public FilePerms As FtpPermissions
```

[C#]

```
public FtpPermissions FilePerms;
```

Remarks

For those familiar with UNIX, the file permissions are the same as those used by the **chmod** command. For the proprietary Sterling directory formats, a bit map representing the status codes and transfer protocol of the file are stored in this member.

See Also

FtpClient.FileInformation Class | SocketTools Namespace

FtpClient.FileInformation.FileSize Field

Specifies the size of the file in bytes.

[Visual Basic]

```
Public FileSize As Long
```

[C#]

```
public long FileSize;
```

See Also

FtpClient.FileInformation Class | SocketTools Namespace

FtpClient.FileInformation.FileVersion Field

Specifies the number of revisions made to the file.

[Visual Basic]

```
Public FileVersion As Integer
```

[C#]

```
public int FileVersion;
```

See Also

FtpClient.FileInformation Class | SocketTools Namespace

FtpClient.FileInformation.IsDirectory Field

Specifies if the file is a directory or regular file.

[Visual Basic]

```
Public IsDirectory As Boolean
```

[C#]

```
public bool IsDirectory;
```

See Also

FtpClient.FileInformation Class | SocketTools Namespace

FtpClient.PortRange Structure

This structure is used by the ActivePorts property which allows the client to specify the port range used for active mode file transfers.

For a list of all members of this type, see FtpClient.PortRange Members.

System.Object
  System.ValueType
    SocketTools.FtpClient.PortRange

[Visual Basic]
```
Public Structure FtpClient.PortRange
```

[C#]
```
public struct FtpClient.PortRange
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

FtpClient.PortRange Members | SocketTools Namespace

FtpClient.PortRange Members

FtpClient.PortRange overview

Public Instance Constructors

FtpClient.PortRange Constructor	Initializes a new instance of the **PortRange** structure with a specified range of port numbers.

Public Instance Fields

HighPort	Specifies the high port number used for active file transfers.
LowPort	Specifies the low port number used for active file transfers.

Public Instance Methods

Equals (inherited from ValueType)	Indicates whether this instance and a specified object are equal.
GetHashCode (inherited from ValueType)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from ValueType)	Returns the fully qualified type name of this instance.

See Also

FtpClient.PortRange Class | SocketTools Namespace

FtpClient.PortRange Constructor

Initializes a new instance of the **PortRange** structure with a specified range of port numbers.

[Visual Basic]
```
Public Sub New( _
   ByVal lowPort As Integer, _
   ByVal highPort As Integer _
)
```

[C#]
```
public FtpClient.PortRange(
   int lowPort,
   int highPort
);
```

Parameters

lowPort
An integer value which specifies the low port number.

highPort
An integer value which specifies the high port number.

Return Value

The minimum port number is 1025 and the maximum port number is 65535. Port numbers outside of this range of values will be silently adjusted.

See Also

FtpClient.PortRange Class | SocketTools Namespace

FtpClient.PortRange Fields

The fields of the **FtpClient.PortRange** structure are listed below. For a complete list of **FtpClient.PortRange** structure members, see the FtpClient.PortRange Members topic.

Public Instance Fields

◆ HighPort	Specifies the high port number used for active file transfers.
◆ LowPort	Specifies the low port number used for active file transfers.

See Also

FtpClient.PortRange Class | SocketTools Namespace

FtpClient.PortRange.HighPort Field

Specifies the high port number used for active file transfers.

[Visual Basic]

```
Public HighPort As Integer
```

[C#]

```
public int HighPort;
```

Remarks

While this member may be assigned to any integer value, the port range is normalized and the maximum port number that may be used is 65535.

See Also

FtpClient.PortRange Class | SocketTools Namespace

FtpClient.PortRange.LowPort Field

Specifies the low port number used for active file transfers.

[Visual Basic]
```
Public LowPort As Integer
```

[C#]
```
public int LowPort;
```

Remarks

While this member may be assigned to any integer value, the port range is normalized and the minimum port number that may be used is 1025.

See Also

FtpClient.PortRange Class | SocketTools Namespace

FtpClient.SecureChannel Structure

The structure used to modify or return the current mode for the secure command or data channel.

For a list of all members of this type, see FtpClient.SecureChannel Members.

System.Object
  System.ValueType
    SocketTools.FtpClient.SecureChannel

[Visual Basic]
```
Public Structure FtpClient.SecureChannel
```

[C#]
```
public struct FtpClient.SecureChannel
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The **SecureChannel** structure specifies the current channel modes for the client session. This structure is used with the **ChannelMode** property when a secure connection is established with the server.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

FtpClient.SecureChannel Members | SocketTools Namespace

FtpClient.SecureChannel Members

FtpClient.SecureChannel overview

Public Instance Constructors

FtpClient.SecureChannel Constructor	Initializes a new instance of the **SecureChannel** structure with the specified values.

Public Instance Fields

Command	Specifies the secure channel mode used when sending commands to the server.
Data	Specifies the secure channel mode used when uploading or downloading files.

Public Instance Methods

Equals (inherited from ValueType)	Indicates whether this instance and a specified object are equal.
GetHashCode (inherited from ValueType)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from ValueType)	Returns the fully qualified type name of this instance.

See Also

FtpClient.SecureChannel Class | SocketTools Namespace

FtpClient.SecureChannel Constructor

Initializes a new instance of the **SecureChannel** structure with the specified values.

[Visual Basic]

```
Public Sub New( _
   ByVal commandMode As FtpChannelMode, _
   ByVal dataMode As FtpChannelMode _
)
```

[C#]

```
public FtpClient.SecureChannel(
   FtpChannelMode commandMode,
   FtpChannelMode dataMode
);
```

Parameters

commandMode
A FtpChannelMode enumeration which specifies the channel mode used for commands sent to the server.

dataMode
A FtpChannelMode enumeration which specifies the channel mode used for file transfers.

See Also

FtpClient.SecureChannel Class | SocketTools Namespace

FtpClient.SecureChannel Fields

The fields of the **FtpClient.SecureChannel** structure are listed below. For a complete list of **FtpClient.SecureChannel** structure members, see the FtpClient.SecureChannel Members topic.

Public Instance Fields

Command	Specifies the secure channel mode used when sending commands to the server.
Data	Specifies the secure channel mode used when uploading or downloading files.

See Also

FtpClient.SecureChannel Class | SocketTools Namespace

FtpClient.SecureChannel.Command Field

Specifies the secure channel mode used when sending commands to the server.

[Visual Basic]

```
Public Command As FtpChannelMode
```

[C#]

```
public FtpChannelMode Command;
```

See Also

FtpClient.SecureChannel Class | SocketTools Namespace

FtpClient.SecureChannel.Data Field

Specifies the secure channel mode used when uploading or downloading files.

[Visual Basic]

```
Public Data As FtpChannelMode
```

[C#]

```
public FtpChannelMode Data;
```

See Also

FtpClient.SecureChannel Class | SocketTools Namespace

FtpClient Properties

The properties of the **FtpClient** class are listed below. For a complete list of **FtpClient** class members, see the FtpClient Members topic.

Public Instance Properties

Account	Get or sets a value that specifies the account name for the current user.
ActivePorts	Gets and sets the port numbers used for active mode file transfers.
AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
Blocking	Gets and sets a value which indicates if the client is in blocking mode.
BufferSize	Gets and sets the size of the internal send and receive buffer that will be used during data transfers.
CertificateExpires	Get a value that specifies the date that the security certificate expires.
CertificateIssued	Get a value that specifies the date that the security certificate was issued.
CertificateIssuer	Get a value that provides information about the organization that issued the certificate.
CertificateName	Gets and sets a value that specifies the name of the client certificate.
CertificatePassword	Gets and sets the password associated with the client certificate.
CertificateStatus	Gets a value which indicates the status of the security certificate returned by the remote host.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateSubject	Gets a value that provides information about the organization that the server certificate was issued to.
CertificateUser	Gets and sets the user that owns the client certificate.
ChannelMode	Set or return the security mode for the specified communications channel.
CipherStrength	Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.
DirectoryFormat	Gets and sets a value which specifies the current directory format type.

Encoding	Gets and sets the character encoding that is used when a file name is sent to the server.
Features	Gets and sets the features that are currently enabled for the current session.
FileMask	Gets and sets the value which specifies the default wildcard file mask.
FileType	Gets and sets a value which specifies the type of file that is being transferred.
Fingerprint	Gets a value that can be used to uniquely identify the server.
Handle	Gets a value that specifies the client handle allocated for the current session.
HashStrength	Gets a value which specifies the length of the message digest that was selected for a secure connection.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsConnected	Gets a value which indicates if a connection to the remote host has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsReadable	Gets a value which indicates if there is data available to be read from the socket connection to the server.
IsWritable	Gets a value which indicates if data can be written to the client without blocking.
KeepAlive	Gets and sets a Boolean value which specifies if the client should attempt to keep the server connection alive.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets the local Internet address that the client is bound to.
Localize	Gets and sets a value which specifies if time and dates should be adjusted for the current timezone.
LocalName	Gets a value which specifies the host name for the local system.

LocalPort	Gets the local port number the client is bound to.
Options	Gets and sets a value which specifies one or more client options.
ParseList	Gets and sets a value that specifies if directory listings should be automatically parsed
Passive	Gets and sets a value which specifies if passive mode file transfers should be enabled.
Password	Gets and sets the password used to authenticate the client session.
Priority	Gets and sets a value which specifies the priority of file transfers.
ProxyHost	Gets and sets the hostname or IP address of a proxy server.
ProxyPassword	Gets and sets the password used to authenticate the connection to a proxy server.
ProxyPort	Gets and sets a value that specifies the proxy server port number.
ProxyType	Gets and sets the type of proxy server the client will use to establish a connection.
ProxyUser	Gets and sets the username used to authenticate the connection to a proxy server.
RemoteFile	Gets and sets the file name specified in the current URL.
RemotePath	Gets and sets the path specified in the current URL.
RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.
ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.
Secure	Gets and sets a value which specifies if a secure connection is established.
SecureCipher	Gets a value that specifies the encryption algorithm used for a secure connection.
SecureHash	Gets a value that specifies the message digest algorithm used for a secure connection.
SecureKeyExchange	Gets a value that specifies the key exchange algorithm used for a secure connection.
SecureProtocol	Gets and sets a value which specifies the protocol

	used for a secure connection.
Status	Gets a value which specifies the current status of the client.
System	Gets a string value which identifies the server.
TaskCount	Get the number of active background file transfers.
TaskId	Get the task identifier for the last background file transfer.
TaskList	Get an array of active background task identifiers.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
TimeZone	Gets and sets the current timezone offset in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
TransferBytes	Gets a value which specifies the number of bytes transferred to or from the remote server.
TransferRate	Gets a value which specifies the data transfer rate in bytes per second.
TransferTime	Gets a value which specifies the number of seconds elapsed during a data transfer.
URL	Gets and sets the current URL used to access a file on the server.
UserName	Gets and sets the username used to authenticate the client session.
Version	Gets a value which returns the current version of the FtpClient class library.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Account Property

Get or sets a value that specifies the account name for the current user.

[Visual Basic]

```
Public Property Account As String
```

[C#]

```
public string Account {get; set;}
```

Property Value

A string which specifies the account name for the current user.

Remarks

The **Account** property specifies the account name of the current user, if it is required by the server for authentication. Not all servers require an account name, in which case this property is ignored.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.ActivePorts Property

Gets and sets the port numbers used for active mode file transfers.

[Visual Basic]

```
Public Property ActivePorts As PortRange
```

[C#]

```
public FtpClient.PortRange ActivePorts {get; set;}
```

Property Value

A **PortRange** structure which specifies the minimum and maximum port numbers used for active mode file transfers.

Remarks

This property is used to change the default port numbers used for active mode file transfers. Active mode is used when the **Passive** property is set to false. Instead of the client establishing an outbound connection to the server for the file transfer, it listens for an inbound connection from the server back to the client. In most cases, passive mode transfers are preferred because they mitigate potential compatibility issues with firewalls and NAT routers.

If active mode transfers are required, the default port range used when listening for the server connection is between 1024 and 5000. This is the standard range of ephemeral ports used by the Windows operating system. However, under some circumstances that range of ports may be too small, or a firewall may be configured to deny inbound connections on ephemeral ports. In that case, the **ActivePorts** property can be used to specify a different range of port numbers.

While it is technically permissible to assign the low and high port numbers to the same value, effectively specifying a single active port number, this is not recommended as it can cause the transfer to fail unexpectedly if multiple file transfers are performed. A minimum range of at least 1000 ports is recommended. For example, if you specify a low port value of 40000 then it is recommended that the high port value be at least 41000. The maximum port value is 65535.

See Also

FtpClient Class | SocketTools Namespace | PortRange Structure

FtpClient.AutoResolve Property

Gets and sets a value that determines if host names and addresses are automatically resolved.

[Visual Basic]

```
Public Property AutoResolve As Boolean
```

[C#]

```
public bool AutoResolve {get; set;}
```

Property Value

Returns **true** if host names are automatically resolved to Internet addresses. The default value is **false**.

Remarks

Setting the **AutoResolve** property determines if the class automatically resolves host names and addresses specified by the **HostName** and **HostAddress** properties. If set to **true**, setting the **HostName** property will cause the class to automatically determine the corresponding IP address and update the **HostAddress** property accordingly. Likewise, setting the **HostAddress** property will cause the class to determine the host name and update the **HostName** property. Setting this property to **false** prevents the class from resolving host names until a connection attempt is made.

It is important to note that setting the **HostName** or **HostAddress** property may cause the current thread to block, sometimes for several seconds, until the name or address is resolved. To prevent this behavior, set this property value to **false**.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Blocking Property

Gets and sets a value which indicates if the client is in blocking mode.

[Visual Basic]

```
Public Property Blocking As Boolean
```

[C#]

```
public bool Blocking {get; set;}
```

Property Value

Returns **true** if the client is in blocking mode; otherwise it returns **false**. The default value is **true**.

Remarks

Setting the **Blocking** property determines if client operations complete synchronously or asynchronously. If set to **true**, then each client operation (such as sending or receiving data) will return when the operation has completed or timed-out. If set to **false**, client operations will return immediately. If the operation would result in the client blocking (such as attempting to read data when no data has been sent by the remote host), an error is generated.

It is important to note that certain events, such as **OnDisconnect, OnRead** and **OnWrite** are only fired if the client is in non-blocking mode.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.BufferSize Property

Gets and sets the size of the internal send and receive buffer that will be used during data transfers.

[Visual Basic]

```
Public Property BufferSize As Integer
```

[C#]

```
public int BufferSize {get; set;}
```

Property Value

An integer value which specifies the size of the internal transfer buffer, expressed in bytes.

Remarks

Setting the **BufferSize** property specifies the size in bytes of an internal buffer that will be used during data transfers. Any set value greater than or equal to zero is acceptable. If the value is set to zero, then the default value of 4096 will be used. If the set value is between 1 and 255, inclusive, the buffer size will be set to 256. The maximum value is 1048576.

The speed of data transfers, particularly on uploads, may be sensitive to network type and configuration, and the size of the internal buffer used for data transfers. The default size of this buffer will result in good performance for a wide range of network characteristics. A larger buffer will not necessarily result in better performance. For example, a multiple of 1460, which is the typical Maximum Transmission Unit (MTU), may be optimal in many situations.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.CertificateExpires Property

Get a value that specifies the date that the security certificate expires.

[Visual Basic]

```
Public ReadOnly Property CertificateExpires As String
```

[C#]

```
public string CertificateExpires {get;}
```

Property Value

A string which specifies a date using the local date and time format.

Remarks

The **CertificateExpires** property returns a string that specifies the date and time that the security certificate expires. This property will return an empty string if a secure connection has not been established with the remote host.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.CertificateIssued Property

Get a value that specifies the date that the security certificate was issued.

[Visual Basic]
```
Public ReadOnly Property CertificateIssued As String
```

[C#]
```
public string CertificateIssued {get;}
```

Property Value

A string which specifies a date using the local date and time format.

Remarks

The **CertificateIssued** property returns a string that specifies the date and time that the security certificate was issued. This property will return an empty string if a secure connection has not been established with the remote host.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.CertificateIssuer Property

Get a value that provides information about the organization that issued the certificate.

[Visual Basic]
```
Public ReadOnly Property CertificateIssuer As String
```

[C#]
```
public string CertificateIssuer {get;}
```

Property Value

A string that contains a comma separated list of name value pairs.

Remarks

The **CertificateIssuer** property returns a string that contains information about the organization that issued the server certificate. The string value is a comma separated list of tagged name and value pairs. In the nomenclature of the X.500 standard, each of these pairs are called a relative distinguished name (RDN), and when concatenated together, forms the issuer's distinguished name (DN). For example:

```
C=US, O="RSA Data Security, Inc.", OU=Secure Server Certification Authority
```

To obtain a specific value, such as the name of the issuer or the issuer's country, the application must parse the string returned by this property. Some of the common tokens used in the distinguished name are:

Token	Description
C	The ISO standard two character country code.
S	The name of the state or province.
L	The name of the city or locality.
O	The name of the company or organization.
OU	The name of the department or organizational unit
CN	The common name; with X.509 certificates, this is the domain name of the site the certificate was issued for.

This property will return an empty string if a secure connection has not been established with the remote host.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.CertificateName Property

Gets and sets a value that specifies the name of the client certificate.

[Visual Basic]

```
Public Property CertificateName As String
```

[C#]

```
public string CertificateName {get; set;}
```

Property Value

A string which specifies the certificate name.

Remarks

The **CertificateName** property is used to specify the name of a client certificate to use when establishing a secure connection. It is only required that you set this property value if the server requires a client certificate for authentication. If this property is not set, a client certificate will not be provided to the server. If a certificate name is specified, the certificate must have a private key associated with it, otherwise the connection attempt will fail because the control will be unable to create a security context for the session.

When the certificate store is searched for a matching certificate, it will first search for any certificate with a friendly name that matches the property value. If no valid certificate is found, it will then search for a certificate with a matching common name.

Certificates may be installed and viewed on the local system using the Certificate Manager that is included with the Windows operating system. For more information, refer to the documentation for the Microsoft Management Console.

See Also

FtpClient Class | SocketTools Namespace | CertificateStore Property | Secure Property

FtpClient.CertificateStatus Property

Gets a value which indicates the status of the security certificate returned by the remote host.

[Visual Basic]

```
Public ReadOnly Property CertificateStatus As SecurityCertificate
```

[C#]

```
public FtpClient.SecurityCertificate CertificateStatus {get;}
```

Property Value

A SecurityCertificate enumeration value which specifies the status of the certificate.

Remarks

The **CertificateStatus** property is used to determine the status of the security certificate returned by the remote host when a secure connection has been established. This property value should be checked after the connection to the server has completed, but prior to beginning a transaction.

Note that if the certificate cannot be validated, the secure connection will not be automatically terminated. It is the responsibility of your application to determine the best course of action to take if the certificate is invalid. Even if the security certificate cannot be validated, the data exchanged with the remote host will still be encrypted.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.CertificateStore Property

Gets and sets a value that specifies the name of the local certificate store.

[Visual Basic]

```
Public Property CertificateStore As String
```

[C#]

```
public string CertificateStore {get; set;}
```

Property Value

A string which specifies the certificate store name. The default value is the current user's personal certificate store.

Remarks

The **CertificateStore** property is used to specify the name of the certificate store which contains the security certificate to use when establishing a secure connection. The certificate may either be stored in the registry or in a file. If the certificate is stored in the registry, then this property should be set to one of the following predefined values:

Store Name	Description
CA	Certification authority certificates. These are certificates that are issued by entities which are entrusted to issue certificates to other individuals or organizations. Companies such as VeriSign and Thawte act as certification authorities.
MY	Personal certificates and their associated private keys for the current user. This store typically holds the client certificates used to establish a user's credentials. If a certificate store is not specified, this is the default value that is used.
ROOT	Certificates that have been self-signed by a certificate authority. Root certificates for a number of different certification authorities such as VeriSign and Thawte are installed as part of the operating system and periodically updated by Microsoft.

In most cases the client certificate will be installed in the user's personal certificate store, and therefore it is not necessary to set this property value because that is the default location that will be used to search for the certificate. This property is only used if the **CertificateName** property is also set to a valid certificate name.

If you are using a local certificate store, with the certificate and private key stored in the registry, you can explicitly specify whether the certificate store for the current user or the local machine (all users) should be used. This is done by prefixing the certificate store name with "HKCU" for the current user, or "HKLM" for the local machine. For example, a certificate store name of "HKLM:MY" would specify the personal certificate store for the local machine, rather than the current user. If neither prefix is specified, it will default to the certificate store for the current user.

This property may also be used to specify a file that contains the client certificate. In this case, the property should specify the full path to the file and must contain both the certificate and private key in PKCS12

format. If the file is protected by a password, the **CertificatePassword** property must also be set to specify the password.

See Also

FtpClient Class | SocketTools Namespace | CertificatePassword Property | Secure Property

FtpClient.CertificateSubject Property

Gets a value that provides information about the organization that the server certificate was issued to.

[Visual Basic]

```
Public ReadOnly Property CertificateSubject As String
```

[C#]

```
public string CertificateSubject {get;}
```

Property Value

A string that contains a comma separated list of name value pairs.

Remarks

The **CertificateSubject** property returns a string that contains information about the organization that the server certificate was issued to. The string value is a comma separated list of tagged name and value pairs. In the nomenclature of the X.500 standard, each of these pairs are called a relative distinguished name (RDN), and when concatenated together, forms the issuer's distinguished name (DN). For example:

```
C=US, O="RSA Data Security, Inc.", OU=Secure Server Certification Authority
```

To obtain a specific value, such as the name of the issuer or the issuer's country, the application must parse the string returned by this property. Some of the common tokens used in the distinguished name are:

Token	Description
C	The ISO standard two character country code.
S	The name of the state or province.
L	The name of the city or locality.
O	The name of the company or organization.
OU	The name of the department or organizational unit
CN	The common name; with X.509 certificates, this is the domain name of the site the certificate was issued for.

This property will return an empty string if a secure connection has not been established with the remote host.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.ChannelMode Property

Set or return the security mode for the specified communications channel.

[Visual Basic]

```
Public Property ChannelMode As SecureChannel
```

[C#]

```
public FtpClient.SecureChannel ChannelMode {get; set;}
```

Property Value

An FtpChannelMode enumeration value which specifies the current channel mode.

Remarks

The **ChannelMode** property is used to change the default mode for the specified channel, and is typically used to control whether or not data is encrypted during a file transfer. If a standard, non-secure connection has been established with the server, an error will be returned if you specify the **channelSecure** mode for either channel.

If you have established a secure connection and then specify the **channelClear** mode for the command channel, the client will send the CCC command to the server to indicate that commands should no longer be encrypted. If the server does not support this command, an error will be returned and the channel mode will remain unchanged. Once the command channel has been changed to clear mode, it cannot be changed back to secure mode. You must disconnect and re-connect to the server if you want to resume sending commands over an encrypted channel.

Changing the mode for the data channel requires that the server support the PROT command. If this command is not supported by the server, an exception will be thrown which must be handled by the application. You can only set a channel to secure mode if the **Secure** property is also set to **true**.

It is important to note that this property should only be used after a connection has been established with the server. If you attempt to read the property or change a value prior to calling the **Connect** method, an exception will be thrown.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.CipherStrength Property

Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.

[Visual Basic]
```
Public ReadOnly Property CipherStrength As Integer
```

[C#]
```
public int CipherStrength {get;}
```

Property Value

An integer value which specifies the encryption key length if a secure connection has been established; otherwise a value of 0 is returned.

Remarks

The **CipherStrength** property returns the number of bits in the key used to encrypt the secure data stream. Common values returned by this property are 128 and 256. A key length of 40 or 56 bits is considered insecure and subject to brute force attacks. 128-bit and 256-bit keys are considered secure. If this property returns a value of 0, this means that a secure connection has not been established with the remote host.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.DirectoryFormat Property

Gets and sets a value which specifies the current directory format type.

[Visual Basic]

```
Public Property DirectoryFormat As FtpDirectoryFormat
```

[C#]

```
public FtpClient.FtpDirectoryFormat DirectoryFormat {get; set;}
```

Property Value

An FtpDirectoryFormat enumeration value which specifies the current directory format.

Remarks

This property should only be set if the client cannot automatically determine the directory format returned by the server. The default directory format is determined both by the server's operating system and by analyzing the format of the data returned by the server. If the class is unable to automatically determine the format, it will attempt to parse the list of files as though it is a UNIX style listing.

If this property is set to the default value **FtpDirectoryFormat.formatAuto** and the class can determine from the format of the file listing returned by the server, then the property will change value upon the first call to the **GetFirstFile** method.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Features Property

Gets and sets the features that are currently enabled for the current session.

[Visual Basic]

```
Public Property Features As FtpFeatures
```

[C#]

```
public FtpClient.FtpFeatures Features {get; set;}
```

Property Value

An FtpFeatures enumeration which specifies the features that are available for the current client session.

Remarks

When a client connection is first established, all features are enabled by default. However, as the client issues commands to the server, if the server reports that the command is unrecognized that feature will automatically be disabled in the client.

For example, the first time an application calls the **GetFileSize** method to determine the size of a file, the control will try to use the SIZE command. If the server reports that the SIZE command is not available, that feature will be disabled and the control will not use the command again during the session unless it is explicitly re-enabled. This is designed to prevent the control from repeatedly sending invalid commands to a server, which may result in the server aborting the connection.

Setting the **Features** property enables those features which have been specified. More than one feature may be enabled by combining the above constants using a bitwise Or operator. To test if a particular feature has been enabled, use the bitwise And operator.

Because features are specific to the current session, once you disconnect from the server they are reset. Even if you wish to reconnect to the same server, you must explicitly set the **Features** property again to those features which you wish to enable. Setting the **Features** property when the control is not connected to a server will cause the client session to only use those specified features for the next connection that is established. Setting the **Features** property during an active connection will change the features available for that session.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.FileMask Property

Gets and sets the value which specifies the default wildcard file mask.

[Visual Basic]

```
Public Property FileMask As String
```

[C#]

```
public string FileMask {get; set;}
```

Property Value

A string which specifies the current wildcard file mask.

Remarks

The **FileMask** property specifies the default wildcard mask to be used when uploading or downloading multiple files. The default value of an empty string indicates that all files in the specified directory should be uploaded or downloaded. Typically, this property is set to a wildcard mask that limits the files downloaded from the server to those which match a specific extension. For example, to download only those files that end in a ".dat" extension, the property could be set to the value "*.dat"

Note that the type of wildcards which may be used depend on the server and the type of file system that it is using. Take particular care when dealing with file systems that distinguish between upper- and lower-case letters in a filename.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.FileType Property

Gets and sets a value which specifies the type of file that is being transferred.

[Visual Basic]
```
Public Property FileType As FtpFileType
```

[C#]
```
public FtpClient.FtpFileType FileType {get; set;}
```

Property Value

An **FtpFileType** enumeration which specifies the type of file being uploaded or downloaded.

Remarks

The file type should be set before a file is uploaded or downloaded from the remote server. Once the file type is set, it is in effect for all files that are subsequently transferred.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Fingerprint Property

Gets a value that can be used to uniquely identify the server.

[Visual Basic]

```
Public ReadOnly Property Fingerprint As String
```

[C#]

```
public string Fingerprint {get;}
```

Property Value

A string value that can be used to uniquely identify a server when the SSH protocol is used.

Remarks

The **Fingerprint** property returns a string that consists of a series of hexadecimal values separated by colons. The value is unique to the server, and is an MD5 hash of the RSA host key. An application can use this value to determine if a connection has been established with the server previously by storing the server's host name, IP address and fingerprint in a file, registry key or a database.

Note that this property only returns a meaningful value after a secure connection has been established using the SSH protocol. For all other connections, it will return an empty string.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Handle Property

Gets a value that specifies the client handle allocated for the current session.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer which represents a client handle. If there is no active connection, a value of -1 is returned.

Remarks

The **Handle** property specifies the numeric descriptor of the current client session and is typically used for debugging or diagnostic purposes.

In SocketTools, handles are used to identify client sessions. A session begins when an instance of the class is used to establish a connection with the server and ends when that connection is terminated. The client handle is defined as an integer type and is used internally to reference the active session. When the connection is terminated, the handle is released, along with any system resources that were allocated for it. An unused handle is identified by the value -1.

It is important to note that the handles returned by this property are not socket handles and cannot be used interchangeably with other objects or Windows API functions. The actual value of the handle is only unique while the client session is active and handle values may be reused. An application should never depend on the **Handle** property returning a specific value.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.HashStrength Property

Gets a value which specifies the length of the message digest that was selected for a secure connection.

[Visual Basic]

```
Public ReadOnly Property HashStrength As Integer
```

[C#]

```
public int HashStrength {get;}
```

Property Value

An integer value which specifies the length of the message digest if a secure connection has been established; otherwise a value of 0 is returned.

Remarks

The **HashStrength** property returns the number of bits used in the message digest (hash) that was selected. Common values returned by this property are 128 and 160. If this property returns a value of 0, this means that a secure connection has not been established with the remote host.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.HostAddress Property

Gets and sets a value which specifies the Internet address used to establish a connection.

[Visual Basic]

```
Public Property HostAddress As String
```

[C#]

```
public string HostAddress {get; set;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **HostAddress** property can be used to set the Internet address for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the address is assigned to a valid host name, the **HostName** property will be updated with that value.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.HostName Property

Gets and sets a value which specifies the host name used to establish a connection.

[Visual Basic]

```
Public Property HostName As String
```

[C#]

```
public string HostName {get; set;}
```

Property Value

A string which specifies a host name.

Remarks

The **HostName** property can be used to set the host name for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the name can be resolved to a valid Internet address, the **HostAddress** property will be updated with that value.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.IsBlocked Property

Gets a value which indicates if the current thread is performing a blocking client operation.

[Visual Basic]

```
Public ReadOnly Property IsBlocked As Boolean
```

[C#]

```
public bool IsBlocked {get;}
```

Property Value

Returns **true** if the current thread is blocking, otherwise returns **false**.

Remarks

The **IsBlocked** property returns **true** if the current thread is blocked performing an operation. Because the Windows Sockets API only permits one blocking operation per thread of execution, this property should be checked before starting any blocking operation in response to an event.

If the **IsBlocked** property returns **false**, this means there are no blocking operations on the current thread at that time. However, this does not guarantee that the next client operation will not fail. An application should always check the return value from a client operation and check the value of the **LastError** property if an error occurs.

Note that this property will return **true** if there is any blocking operation being performed by the current thread, regardless of whether this specific instance of the class is responsible for the blocking operation or not.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.IsConnected Property

Gets a value which indicates if a connection to the remote host has been established.

[Visual Basic]

```
Public ReadOnly Property IsConnected As Boolean
```

[C#]

```
public bool IsConnected {get;}
```

Property Value

Returns **true** if the connection has been established; otherwise returns **false**.

Remarks

The **IsConnected** property can only be used to indicate if there is still a logical connection to the remote host. It cannot be used to detect abnormal conditions such as the remote host aborting the connection, the physical network connection being lost or other critical errors.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.IsInitialized Property

Gets a value which indicates if the current instance of the class has been initialized successfully.

[Visual Basic]
```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]
```
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the class instance has been initialized; otherwise returns **false**.

Remarks

The **IsInitialized** property is used to determine if the current instance of the class has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the class may not be able to initialize itself.

The most common reasons that a class instance may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required networking libraries or not being able to access the system registry.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.IsReadable Property

Gets a value which indicates if there is data available to be read from the socket connection to the server.

[Visual Basic]
```
Public ReadOnly Property IsReadable As Boolean
```

[C#]
```
public bool IsReadable {get;}
```

Property Value

Returns **true** if there is data available to be read; otherwise returns **false**.

Remarks

The **IsReadable** property returns **true** if data can be read from the client without blocking. For non-blocking sessions, this property can be checked before the application attempts to read the client. Note that even if this property does return **true** indicating that there is data available to be read, applications should always check the return value from the **Read** method.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.IsWritable Property

Gets a value which indicates if data can be written to the client without blocking.

[Visual Basic]
```
Public ReadOnly Property IsWritable As Boolean
```

[C#]
```
public bool IsWritable {get;}
```

Property Value

Returns **true** if data can be written to the client; otherwise returns **false**.

Remarks

The **IsWritable** property returns **true** if data can be written to the client without blocking. For non-blocking sessions, this property can be checked before the application attempts to write data to the client. Note that even if this property does return **true** indicating that data can be written to the client, applications should always check the return value from the **Write** method.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.KeepAlive Property

Gets and sets a Boolean value which specifies if the client should attempt to keep the server connection alive.

[Visual Basic]

```
Public Property KeepAlive As Boolean
```

[C#]

```
public bool KeepAlive {get; set;}
```

Property Value

A boolean value which specifies if the client should attempt to maintain the connection with the server over a long period of time.

Remarks

If the **KeepAlive** property is set to **true**, the client will attempt to maintain an active connection to the server over a long period of time. If this property is set to **false**, then no attempt will be made to hold the command channel open.

It is important to note that enabling this option does not guarantee that the connection will be maintained. The application must be written to account for situations where the connection to the server is terminated if it is idle for a long period of time, regardless of the value of this property.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As Integer
```

[C#]

```
public int LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.LocalAddress Property

Gets the local Internet address that the client is bound to.

[Visual Basic]

```
Public ReadOnly Property LocalAddress As String
```

[C#]

```
public string LocalAddress {get;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **LocalAddress** property returns the local Internet address that the client is bound to when a connection is established with a remote host. This property may return either an IPv4 or IPv6 formatted address, depending on the type of connection that was established.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Localize Property

Gets and sets a value which specifies if time and dates should be adjusted for the current timezone.

[Visual Basic]

```
Public Property Localize As Boolean
```

[C#]

```
public bool Localize {get; set;}
```

Property Value

A boolean value which specifies if file time and dates should be adjusted for the local timezone. The default value is false.

Remarks

The **Localize** property controls how remote file date and time values are localized when the **GetFileTime** method is called. If the property is set to **true** the file date and time will be adjusted to the current timezone. If the property is set to **false** the file date and time are returned as UTC (Coordinated Universal Time) values.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.LocalName Property

Gets a value which specifies the host name for the local system.

[Visual Basic]

```
Public ReadOnly Property LocalName As String
```

[C#]

```
public string LocalName {get;}
```

Property Value

A string which specifies the local host name.

Remarks

The **LocalName** property returns the fully-qualified host name assigned to the local system. If the system has not been configured with an Internet domain name, then this property will return the NetBIOS name assigned to the local system.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.LocalPort Property

Gets the local port number the client is bound to.

[Visual Basic]
```
Public ReadOnly Property LocalPort As Integer
```

[C#]
```
public int LocalPort {get;}
```

Property Value

An integer value which specifies a port number. The default value is 0.

Remarks

The **LocalPort** property is used to identify the local port number that the client is bound to to when a connection is established with a remote host.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Options Property

Gets and sets a value which specifies one or more client options.

[Visual Basic]

```
Public Property Options As FtpOptions
```

[C#]

```
public FtpClient.FtpOptions Options {get; set;}
```

Property Value

Returns one or more FtpOptions enumeration flags which specify the options for the client. The default value for this property is **optionNone**.

Remarks

The **Options** property specifies one or more default options options which are used when establishing a connection using the **Connect** method.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.ParseList Property

Gets and sets a value that specifies if directory listings should be automatically parsed

[Visual Basic]

```
Public Property ParseList As Boolean
```

[C#]

```
public bool ParseList {get; set;}
```

Property Value

A boolean value which specifies if directory listings should be automatically parsed. If the value is **true**, then the class will attempt to parse directory listings returned by the server. If the value is **false**, raw directory listings will be returned to the client.

Remarks

The **ParseList** property is used to control how remote file lists are processed. If the property is set to **false**, file lists are not parsed and the application is responsible for parsing the list of files returned by the server.

The class recognizes file listings in UNIX, MS-DOS, VMS, Windows and NetWare formats, and will attempt to automatically determine the format that is being returned by the server. If the server does not return file lists in one of these formats, the **ParseList** property should be set to **false**, and the application must parse the file listing itself.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Passive Property

Gets and sets a value which specifies if passive mode file transfers should be enabled.

[Visual Basic]

```
Public Property Passive As Boolean
```

[C#]

```
public bool Passive {get; set;}
```

Property Value

A boolean value which specifies if passive mode file transfers are enabled. If this value is set to **true**, passive mode is enabled. If the value is set to **false**, then passive mode transfers are disabled. The default value is **true**.

Remarks

When the client uploads or downloads a file and the **Passive** property is set to **false**, the server establishes a second connection back to the client which is used to transfer the file data. However, if the local system is behind a firewall or a NAT router, the server may not be able to create the data connection and the transfer will fail. By setting this property to **true**, it forces the client to establish an outbound data connection with the server. It is recommended that most applications use passive mode whenever possible.

Setting this property to **true** is the same as specifying the optionPassive flag when establishing a connection to the server.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Password Property

Gets and sets the password used to authenticate the client session.

[Visual Basic]

```
Public Property Password As String
```

[C#]

```
public string Password {get; set;}
```

Property Value

A string which specifies the password.

Remarks

If a password is not specified when the **Connect** method is called, the value of this property will be used as the default password when establishing a connection with the server.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Priority Property

Gets and sets a value which specifies the priority of file transfers.

[Visual Basic]
```
Public Property Priority As FtpPriority
```

[C#]
```
public FtpClient.FtpPriority Priority {get; set;}
```

Property Value

Returns a FtpPriority enumeration value which specify the current file transfer priority. The default value for this property is **priorityNormal**.

Remarks

The **Priority** property can be used to control the processor usage, memory and network bandwidth allocated for file transfers. The default priority balances resource utilization and transfer speed while ensuring that a single-threaded application remains responsive to the user. Lower priorities reduce the overall resource utilization at the expense of transfer speed. For example, if you create a worker thread to download a file in the background and want to ensure that it has a minimal impact on the process, the **priorityBackground** value can be used.

Higher priority values increase the memory allocated for the transfers and increases processor utilization for the transfer. The **priorityCritical** priority maximizes transfer speed at the expense of system resources. It is not recommended that you increase the file transfer priority unless you understand the implications of doing so and have thoroughly tested your application. If the file transfer is being performed in the main UI thread, increasing the priority may interfere with the normal processing of Windows messages and cause the application to appear to become non-responsive. It is also important to note that when the priority is set to **priorityCritical**, normal progress events will not be generated during the transfer.

See Also

FtpClient Class | SocketTools Namespace | FtpPriority Enumeration

FtpClient.ProxyHost Property

Gets and sets the hostname or IP address of a proxy server.

[Visual Basic]

```
Public Property ProxyHost As String
```

[C#]

```
public string ProxyHost {get; set;}
```

Property Value

A string which specifies the hostname or IP address of the proxy server that will be used when establishing a connection.

Remarks

The **ProxyHost** property should be set to the name of the proxy server that you want to connect to. This property may be set to either a fully qualified domain name, or an IP address. This property is only used if the **ProxyType** property specifies a proxy server type other than **proxyNone**.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.ProxyPassword Property

Gets and sets the password used to authenticate the connection to a proxy server.

[Visual Basic]

```
Public Property ProxyPassword As String
```

[C#]

```
public string ProxyPassword {get; set;}
```

Property Value

A string which specifies a password.

Remarks

The **ProxyPassword** property specifies the password used to authenticate the user to the proxy server. If a password is not required by the server, this property is ignored.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.ProxyPort Property

Gets and sets a value that specifies the proxy server port number.

[Visual Basic]

```
Public Property ProxyPort As Integer
```

[C#]

```
public int ProxyPort {get; set;}
```

Property Value

An integer value which specifies the proxy port number.

Remarks

The **ProxyPort** property is used to set the port number that the control will use to establish a connection with the proxy server. A value of zero specifies that the client will connect to the proxy server using the standard FTP service port.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.ProxyType Property

Gets and sets the type of proxy server the client will use to establish a connection.

[Visual Basic]

```
Public Property ProxyType As FtpProxyType
```

[C#]

```
public FtpClient.FtpProxyType ProxyType {get; set;}
```

Property Value

An FtpProxyType enumeration which specifies the type of proxy that the client will connect through.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.ProxyUser Property

Gets and sets the username used to authenticate the connection to a proxy server.

[Visual Basic]

```
Public Property ProxyUser As String
```

[C#]

```
public string ProxyUser {get; set;}
```

Property Value

A string which specifies the username.

Remarks

The **ProxyUser** property specifies the user that is logging in to the proxy server. If the proxy server does not require the user to login, then this property is ignored.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.RemoteFile Property

Gets and sets the file name specified in the current URL.

[Visual Basic]
```
Public Property RemoteFile As String
```

[C#]
```
public string RemoteFile {get; set;}
```

Property Value

A string which specifies a file name on the server.

Remarks

The **RemoteFile** property returns the name of the file that was specified when the **URL** property was set. Changing the value of this property causes the current URL to change. The class does not check to make sure that the remote file name is valid or that it actually exists on the server.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.RemotePath Property

Gets and sets the path specified in the current URL.

[Visual Basic]

```
Public Property RemotePath As String
```

[C#]

```
public string RemotePath {get; set;}
```

Property Value

A string which specifies a directory path on the server.

Remarks

The **RemotePath** property returns the path that was specified when the **URL** property was set. Changing the value of this property causes the current URL to change. The class does not check to make sure that the remote path is valid or that it actually exists on the server.

Note that the path is relative to the user's home directory and should not be considered an absolute path from the root directory on the server. If no username and password are provided when the connection is established, then an anonymous session is used and the path is relative to the public directory defined by the FTP server.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.RemotePort Property

Gets and sets a value which specifies the remote port number.

[Visual Basic]

```
Public Property RemotePort As Integer
```

[C#]

```
public int RemotePort {get; set;}
```

Property Value

An integer value which specifies a port number.

Remarks

The **RemotePort** property is used to set the port number that will be used to establish a connection with a remote host. If the port number specifies a well-known port, the **RemoteService** property will be updated with that name.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.ResultCode Property

Gets a value which specifies the last result code returned by the server.

[Visual Basic]

```
Public ReadOnly Property ResultCode As Integer
```

[C#]

```
public int ResultCode {get;}
```

Property Value

An integer value which specifies the last result code returned by the server.

Remarks

Result codes are three-digit numeric values returned by the remote server and may be broken down into the following ranges:

ResultCode	Description
100-199	Positive preliminary result. This indicates that the requested action is being initiated, and the client should expect another reply from the server before proceeding.
200-299	Positive completion result. This indicates that the server has successfully completed the requested action.
300-399	Positive intermediate result. This indicates that the requested action cannot complete until additional information is provided to the server.
400-499	Transient negative completion result. This indicates that the requested action did not take place, but the error condition is temporary and may be attempted again.
500-599	Permanent negative completion result. This indicates that the requested action did not take place.

It is important to note that while some result codes have become standardized, not all servers respond to commands using the same result codes. For example, one server may respond with a result code of 221 to indicate success, while another may respond with a value of 235. It is recommended that applications check for ranges of values to determine if a command was successful, not a specific value.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.ResultString Property

Gets a string value which describes the result of the previous command.

[Visual Basic]

```
Public ReadOnly Property ResultString As String
```

[C#]

```
public string ResultString {get;}
```

Property Value

A string which describes the result of the previous command executed on the server.

Remarks

The **ResultString** property returns the result string from the last action taken by the client. This string is generated by the remote server, and typically is used to describe the result code. For example, if an error is indicated by the result code, the result string may describe the condition that caused the error.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Secure Property

Gets and sets a value which specifies if a secure connection is established.

[Visual Basic]

```
Public Property Secure As Boolean
```

[C#]

```
public bool Secure {get; set;}
```

Property Value

Returns **true** if a secure connection is established; otherwise returns **false**. The default value is **false**.

Remarks

The **Secure** property determines if a secure connection is established with the remote host. The default value for this property is **false**, which specifies that a standard connection to the server is used. To establish a secure connection, the application should set this property value to **true** prior to calling the **Connect** method. Once the connection has been established, the client may exchange data with the server as with standard connections.

It is strongly recommended that any application that sets this property **true** use error handling to trap an errors that may occur. If the class is unable to initialize the security libraries, or otherwise cannot create a secure session for the client, an exception may be generated when this property value is set to **true**.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.SecureCipher Property

Gets a value that specifies the encryption algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureCipher As SecureCipherAlgorithm
```

[C#]

```
public FtpClient.SecureCipherAlgorithm SecureCipher {get;}
```

Property Value

A SecureCipherAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureCipher** property returns a value which identifies the algorithm used to encrypt the data stream. If a secure connection has not been established, this property will return a value of **cipherNone**.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.SecureHash Property

Gets a value that specifies the message digest algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureHash As SecureHashAlgorithm
```

[C#]

```
public FtpClient.SecureHashAlgorithm SecureHash {get;}
```

Property Value

A SecureHashAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureHash** property returns a value which identifies the message digest (hash) algorithm that was selected when a secure connection was established. If a secure connection has not been established, this property will return a value of **hashNone**.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.SecureKeyExchange Property

Gets a value that specifies the key exchange algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureKeyExchange As SecureKeyAlgorithm
```

[C#]

```
public FtpClient.SecureKeyAlgorithm SecureKeyExchange {get;}
```

Property Value

A SecureKeyAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureKeyExchange** property returns a value which identifies the key exchange algorithm that was selected when a secure connection was established. If a secure connection has not been established, this property will return a value of **keyExchangeNone**.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.SecureProtocol Property

Gets and sets a value which specifies the protocol used for a secure connection.

[Visual Basic]
```
Public Property SecureProtocol As SecurityProtocols
```

[C#]
```
public FtpClient.SecurityProtocols SecureProtocol {get; set;}
```

Property Value

A SecurityProtocols enumeration value which identifies the protocol to be used when establishing a secure connection.

Remarks

The **SecureProtocol** property can be used to specify the security protocol to be used when establishing a secure connection with a server or accepting a secure connection from a client. By default, the class will attempt to use either SSL v3 or TLS v1 to establish the connection, with the appropriate protocol automatically selected based on the capabilities of the remote host. It is recommended that you only change this property value if you fully understand the implications of doing so. Assigning a value to this property will override the default protocol and force the class to attempt to use only the protocol specified.

Multiple security protocols may be specified by combining them using a bitwise **or** operator. After a connection has been established, this property will identify the protocol that was selected. Attempting to set this property after a connection has been established will result in an exception being thrown. This property should only be set after setting the **Secure** property to **true** and before calling the **Accept** or **Connect** methods.

In some cases, a server may only accept a secure connection if the TLS v1 protocol is specified. If the security protocol is not compatible with the server, then the connection will fail with an error indicating that the control is unable to establish a security context for the session. In this case, try assigning the property to **protocolTLS1** and attempt the connection again.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Status Property

Gets a value which specifies the current status of the client.

[Visual Basic]

```
Public ReadOnly Property Status As FtpStatus
```

[C#]

```
public FtpClient.FtpStatus Status {get;}
```

Property Value

A FtpStatus enumeration value which specifies the current client status.

Remarks

The **Status** property returns the current status of the client. This property can be used to check on blocking connections to determine if the client is interacting with the remote host before taking some action.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.System Property

Gets a string value which identifies the server.

[Visual Basic]

```
Public ReadOnly Property System As String
```

[C#]

```
public string System {get;}
```

Property Value

A string which identifies the type of server the client has connected to.

Remarks

The **System** property returns information about the remote host operating system. This is a read-only property that can be used by the application to identify the type of server that the client has connected to. Reading this property will cause the SYST command to be sent to the server and will only return a useful value after a connection has been established with the server.

By convention, the first whitespace separated token in the string identifies the general operating system platform. For example, here are some strings commonly returned by various FTP servers:

Examples	Description
UNIX Type: L8	A standard UNIX based server. This is the most common value returned by servers, and this indicates that the server supports UNIX file naming and directory listing conventions. This string may also include additional information such as the specific variant of UNIX and its version. The L8 portion of the string is a convention that lets the client know that a byte consists of 8 bits.
Windows_NT Version 5.0	A standard Windows based server, typically part of Internet Information Services (ISS). The server will use Windows file naming and directory listing conventions. The version identifies the specific release of Windows. For example, version 4.0 specifies Windows NT 4.0 and 5.0 specifies Windows 2000.
VMS V7.1 AlphaServer	A server running the VMS operating system. The server will use the standard file naming and directory listing conventions for that platform. Note that it is possible that a VMS system may also be configured to operate in a UNIX emulation mode, in which case it will return UNIX instead of VMS.
NetWare	A server running the NetWare operating system. The server will use the standard file naming and directory listing conventions for that platform. Note that it is possible that a NetWare system may be configured to operate in a UNIX emulation

	mode, in which case it return UNIX instead of NetWare.
WORLDGROUP Type: L8	A server running the WorldGroup software on the Windows platform. This server supports UNIX file naming and directory listing conventions. WorldGroup is a collaborative workgroup, email and file sharing service which includes an FTP server.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.ThreadModel Property

Gets and sets a value which specifies the threading model for the class instance.

[Visual Basic]
```
Public Property ThreadModel As ThreadingModel
```

[C#]
```
public FtpClient.ThreadingModel ThreadModel {get; set;}
```

Property Value

Returns one or more ThreadingModel enumeration value which specifies the threading model for the client. The default value for this property is **modelSingleThread**.

Remarks

The **ThreadModel** property specifies the threading model that is used by the class instance when a connection is established. The default value for this property is **modelSingleThread**, which specifies that only the thread that established the connection should be permitted to invoke methods. It is important to note that this threading model does not limit the application to a single thread of execution. When a session is established using the **Connect** method, that session is attached to the thread that created it. From that point on, until the session is terminated, only the owner may invoke methods in that instance of the class. The ownership of the class instance may be transferred from one thread to another using the **AttachThread** method.

Setting this property to **modelFreeThread** disables certain internal safety checks that are performed by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads. The application must ensure that no two threads will attempt to invoke a blocking method at the same time. In other words, if one thread invokes a method, the application must ensure that another thread will not attempt to invoke any other method at the same time using the same instance of the class.

Changing the value of this property will not affect an active client session. The threading model must be specified prior to invoking the **Connect** method.

See Also

FtpClient Class | SocketTools Namespace | AttachThread Method | ThreadingModel Enumeration | ThreadModel Attribute

FtpClient.TaskCount Property

Get the number of active background file transfers.

[Visual Basic]

```
Public ReadOnly Property TaskCount As Integer
```

[C#]

```
public int TaskCount {get;}
```

Property Value

An integer value that specifies the number of background file transfers that are currently in progress.

Remarks

The **TaskCount** property returns the number of background file transfers that are currently in progress. One common use for this property is to create a timer that periodically checks this value when a series of background transfers are started. When the property returns a value of zero, that indicates all of the background transfers have completed. This property can also be used to enumerate the active background tasks in conjunction with the **TaskList** property.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.TaskId Property

Get the task identifier for the last background file transfer.

[Visual Basic]
```
Public ReadOnly Property TaskId As Integer
```

[C#]
```
public int TaskId {get;}
```

Property Value

An integer value the uniquely identifies the current background task.

Remarks

The **TaskId** property returns the task ID associated with the current background task. This identifies the last background file transfer that was initiated with a call to the **AsyncGetFile** or **AsyncPutFile** methods. This property value will change with each subsequent background transfer that is performed. If this property returns a value of zero, that indicates that no background tasks have been started for this instance of the class.

To enumerate the active background tasks, use the **TaskCount** property and the **TaskList** array.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.TaskList Property

Get an array of active background task identifiers.

[Visual Basic]

```
Public ReadOnly Property TaskList As ArrayList
```

[C#]

```
public System.Collections.ArrayList TaskList {get;}
```

Property Value

An **ArrayList** object that contains a list of integer values that uniquely identify the active background tasks that have been started by this instance of the class.

Remarks

The **TaskList** property returns a read-only **ArrayList** object that is popularted with the task identifiers for all active background tasks that have been created by this instance of the class. The current number of active tasks can be determined using the **TaskCount** property.

As background tasks complete and additional tasks are started, the values stored in this array will change. The application should never make any assumptions about the numeric values stored in the array or the order they are returned. Task IDs should be considered opaque values that are unique to the process. When a background task completes, its corresponding ID is removed from the list of active tasks and this can potentially change the task ID values associated with each index into the array.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]

```
Public Property ThrowError As Boolean
```

[C#]

```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Timeout Property

Gets and sets a value which specifies a timeout period in seconds.

[Visual Basic]

```
Public Property Timeout As Integer
```

[C#]

```
public int Timeout {get; set;}
```

Property Value

An integer value which specifies a timeout period in seconds.

Remarks

Setting the **Timeout** property specifies the number of seconds until a blocking operation fails and returns an error.

The timeout period is only used when the client is in blocking mode. Although this property can be changed when the client is in non-blocking mode, the value will be ignored until the client is returned to blocking mode.

For most applications it is recommended the timeout period be set between 10 and 20 seconds.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.TimeZone Property

Gets and sets the current timezone offset in seconds.

[Visual Basic]
```
Public Property TimeZone As Integer
```

[C#]
```
public int TimeZone {get; set;}
```

Property Value

An integer value which specifies the current timezone offset in seconds.

Remarks

The **TimeZone** property returns the current offset from UTC in seconds. Setting the property changes the current timezone offset to the specified value. The value of this property is initially determined by the date and time settings on the local system.

This property value is used in conjunction with the **Localize** property to control how date and time localization is handled.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Trace Property

Gets and sets a value which indicates if network function logging is enabled.

[Visual Basic]

```
Public Property Trace As Boolean
```

[C#]

```
public bool Trace {get; set;}
```

Property Value

Returns **true** if network function tracing is enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Trace** property is used to enable (or disable) the tracing of network function calls. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the SocketTools networking classes and is enabled or disabled for an entire process. This means that once trace logging is enabled for a given component, all of the function calls made by the process using any of the SocketTools classes will be logged. For example, if you have an application using both the File Transfer Protocol and Post Office Protocol classes, and you set the **Trace** property to **true**, function calls made by both classes will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all classes used by the process.

If trace logging is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the logfile is fairly large. Since logfiles can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

When redistributing your application, make sure that you include the **SocketTools10.TraceLog.dll** module with your installation. If this library is not present, then no trace output will be generated and the value of the **Trace** property will be ignored. Only those function calls made by the SocketTools networking classes will be logged. Calls made directly to the Windows Sockets API, or calls made by other classes, will not be logged.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.TraceFile Property

Gets and sets a value which specifies the name of the logfile.

[Visual Basic]
```
Public Property TraceFile As String
```

[C#]
```
public string TraceFile {get; set;}
```

Property Value

A string which specifies the name of the file.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named **SocketTools.log** is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFile** property should be set to the same value for each instance. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
 MyApp INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0 MyApp WRN:
connect(46, 192.0.0.1:1234, 16) returned -1 [10035] MyApp ERR: accept(46,
NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the control is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Note that if the specified file cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.TraceFlags Property

Gets and sets a value which specifies the client function tracing flags.

[Visual Basic]

```
Public Property TraceFlags As TraceOptions
```

[C#]

```
public FtpClient.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration which specifies the amount of detail written to the trace logfile.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Because network function tracing is enabled per-process, the trace flags are shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFlags** property should be set to the same value for each instance. Changing the trace flags for any one instance of the class will affect the logging performed for all SocketTools classes used by the application.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being written and the error **errorOperationWouldBlock** occurs, a warning is generated because the application simply needs to attempt to write the data at a later time.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.TransferBytes Property

Gets a value which specifies the number of bytes transferred to or from the remote server.

[Visual Basic]

```
Public ReadOnly Property TransferBytes As Long
```

[C#]

```
public long TransferBytes {get;}
```

Property Value

An integer value which specifies the number of bytes of data transferred to or from the server.

Remarks

The **TransferBytes** property returns the number of bytes that have been copied to or from the remote FTP server. If this property is read while a transfer is ongoing, the property returns the number of bytes that have been copied up to that point. If read after a transfer has completed, the total number of bytes copied is returned. This property value is reset with every data transfer.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.TransferTime Property

Gets a value which specifies the number of seconds elapsed during a data transfer.

[Visual Basic]

```
Public ReadOnly Property TransferTime As Integer
```

[C#]

```
public int TransferTime {get;}
```

Property Value

An integer value which specifies the transfer time in seconds.

Remarks

The **TransferTime** property returns the number of seconds that have elapsed since the data connection was opened on the remote server. If the property is read while a transfer is ongoing, it returns the elapsed time. If the property is read after the transfer is complete, it returns the total number of seconds it took to transfer the data. This property value is reset with every data transfer.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.URL Property

Gets and sets the current URL used to access a file on the server.

[Visual Basic]

```
Public Property URL As String
```

[C#]

```
public string URL {get; set;}
```

Property Value

A string which specifies the current URL.

Remarks

The **URL** property returns the current Uniform Resource Locator string which is used by the control to access a file on the server. URLs have a specific format which provides information about the remote host, port, path and file name, as well as optional information such as a username and password for authentication:

> `ftp://[`*`username`*` : [`*`password`*`] @] remotehost [:`*`remoteport`*`] / [`*`path/`*`...] filename [;`*`type=id`*`]`

The first part of the URL is the protocol and in this case will always be "ftp", or "ftps" if a secure connection is being used. If a username and password is required for authentication, then this will be included in the URL before the name of the remote host; otherwise an anonymous FTP session is assumed. Next, there is the name of the remote host to connect to, optionally followed by a port number. If no port number is given, then the default port for the protocol will be used. This is followed by the path, and then the name of the file on the server. An optional file type may be specified as well, with the type identifier being either "a" for text files or "i" for binary files.

One important consideration when using FTP URLs is that the path is relative to the user's home directory and should not be considered an absolute path from the root directory on the server. If no username and password is provided, then an anonymous session is used and the path is relative to the public directory used by the FTP server.

Here are some typical examples of URLs used to access files on an FTP server:

```
ftp://www.example.com/pub/financial/jan2020.xls
```

> In this example, the remote host is www.example.com, the path is "pub/financial" and the file name is "jan2020.xls". The default port will be used to access the file, and no username and password is provided for authentication so this file must be publicly available to anonymous users.

```
ftp://www.example.com:2121/employees/picnic.doc
```

> In this example, the remote host is www.example.com, the path is "employees" and the file name is "picnic.doc". However, the client should connect to an alternative port number, in this case 2121. This file must also be available to anonymous users because no username or password has been specified.

```
ftps://executive:secret@www.example.com/corporate/projections/sales2020.xls
```

In this example, the remote host is www.example.com and, the path is "corporate/projections" and the file name is "sales2020.xls". Because the protocol is ftps, a secure connection on port 990 will be established. The user name "executive" and password "secret" will be used to authenticate the session.

When setting the **URL** property, the class will parse the string and automatically update the **HostName**, **RemotePort**, **UserName**, **Password**, **RemotePath** and **RemoteFile** properties according to the values specified in the URL. This enables an application to simply provide the URL and then call the **Connect** method to establish the connection.

Note that if this property is assigned a value which cannot be parsed, an exception will be thrown that indicates that the property value is invalid. If the user enters an invalid URL and there is no exception handler, the unhandled exception will terminate the application.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.UserName Property

Gets and sets the username used to authenticate the client session.

[Visual Basic]

```
Public Property UserName As String
```

[C#]

```
public string UserName {get; set;}
```

Property Value

A string which specifies the username.

Remarks

If a username is not specified when the **Connect** method is called, the value of this property will be used as the default username when establishing a connection with the server.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Version Property

Gets a value which returns the current version of the FtpClient class library.

[Visual Basic]

```
Public ReadOnly Property Version As String
```

[C#]

```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the FtpClient class library. This value can be used by an application for validation and debugging purposes.

See Also

FtpClient Class | SocketTools Namespace

FtpClient Methods

The methods of the **FtpClient** class are listed below. For a complete list of **FtpClient** class members, see the FtpClient Members topic.

Public Static (Shared) Methods

ErrorText	Returns the description of an error code.

Public Instance Methods

AddFileType	Associate a file name extension with a specific file type.
AsyncGetFile	Overloaded. Download a file from the server to the local system in the background.
AsyncPutFile	Overloaded. Upload a file from the local system to the server in the background.
AttachThread	Attach an instance of the class to the current thread
Cancel	Cancel the current blocking client operation.
ChangeDirectory	Change the current working directory on the remote server.
CloseDirectory	Close the directory that was previously opened with the OpenDirectory method.
CloseFile	Close the file that was previously opened with the OpenFile method.
Command	Overloaded. Send a custom command to the server.
Connect	Overloaded. Establish a connection with a remote host.
CreateFile	Create a new file or overwrite an existing file.
DeleteFile	Delete a file on the remote server.
Disconnect	Terminate the connection with the remote server.
Dispose	Overloaded. Releases all resources used by FtpClient.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
FileList	Overloaded. Return a list of files on the remote host.
GetData	Overloaded. Transfers the contents of a file on the server and stores it in a byte array.
GetDirectory	Return the current working directory.
GetFile	Overloaded. Download a file from the server to the local system.

GetFileList	Overloaded. Returns an unparsed list of files in the specified directory.
GetFilePermissions	Return the access permissions for a file on the remote system.
GetFileSize	Overloaded. Returns the size of the specified file on the remote server.
GetFileStatus	Returns status information about the specified file.
GetFileTime	Overloaded. Returns the modification date and time for specified file on the remote server.
GetFirstFile	Overloaded. Get information about the first file in a directory listing returned by the server.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetMultipleFiles	Overloaded. Download multiple files from the server to the local system using a wildcard mask.
GetNextFile	Overloaded. Get information about the next file in a directory listing returned by the server.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the FtpClient class.
Login	Overloaded. Login to the remote server.
Logout	Log the current user off the server.
MakeDirectory	Create a new directory on the server.
OpenDirectory	Overloaded. Open the specified directory on the server.
OpenFile	Overloaded. Open an existing file or create a new file on the server.
PutData	Overloaded. Transfers data from a byte array and stores it in a file on the remote server.
PutFile	Overloaded. Upload a file from the local system to the server.
PutMultipleFiles	Overloaded. Upload multiple files from the local system to the server using a wildcard mask.
Read	Overloaded. Read file data from the server and store it in a byte array.
RemoveDirectory	Remove a directory on the server.
RenameFile	Change the name of a file on the server.
Reset	Reset the internal state of the object, resetting all properties to their default values.
SetFilePermissions	Change the access permissions for a file on the

	server.
SetFileTime	Changes the modification date and time for a file on the server.
TaskAbort	Overloaded. Abort the specified asynchronous task.
TaskDone	Overloaded. Determine if an asynchronous task has completed.
TaskResume	Overloaded. Resume execution of an asynchronous task.
TaskSuspend	Overloaded. Suspend execution of an asynchronous task.
TaskWait	Overloaded. Wait for an asynchronous task to complete.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
VerifyFile	Overloaded. Verify that the contents of a file on the local system are the same as the specified file on the server.
Write	Overloaded. Write one or more bytes of data to the server.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the FtpClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.AddFileType Method

Associate a file name extension with a specific file type.

```
[Visual Basic]
Public Function AddFileType( _
   ByVal fileExtension As String, _
   ByVal fileType As FtpFileType _
) As Boolean
```

```
[C#]
public bool AddFileType(
   string fileExtension,
   FtpFileType fileType
);
```

Parameters

fileExtension
A string value which specifies the file name extension.

fileType
A FtpFileType enumeration which specifies the type of file associated with the file extension.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method is used to associate specific file types with file name extensions. The class has an internal list of standard text file extensions which it automatically recognizes. This method can be used to extend or modify that list for the client session.

See Also

FtpClient Class | SocketTools Namespace | FtpFileType Enumeration

FtpClient.AsyncGetFile Method

Download a file from the server to the local system in the background.

Overload List

Download a file from the server to the local system in the background.

public bool AsyncGetFile(string);

Download a file from the server to the local system in the background.

public bool AsyncGetFile(string,string);

Download a file from the server to the local system in the background.

public bool AsyncGetFile(string,string,FtpTransferOptions);

Download a file from the server to the local system in the background.

public bool AsyncGetFile(string,string,FtpTransferOptions,long);

See Also

FtpClient Class | SocketTools Namespace | AsyncPutFile Method

FtpClient.AsyncGetFile Method (String)

Download a file from the server to the local system in the background.

[Visual Basic]

```
Overloads Public Function AsyncGetFile( _
   ByVal localFile As String _
) As Boolean
```

[C#]

```
public bool AsyncGetFile(
   string localFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be created, overwritten or appended to. The file pathing and name conventions must be that of the local host.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncGetFile** method will download the contents of a remote file to a file on the local system. It is similar to the **GetFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being downloaded from the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create a new instance of this class for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is downloaded, the class will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to download more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.AsyncGetFile Overload List | AsyncPutFile Method

FtpClient.AsyncGetFile Method (String, String)

Download a file from the server to the local system in the background.

[Visual Basic]

```
Overloads Public Function AsyncGetFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String _
) As Boolean
```

[C#]

```
public bool AsyncGetFile(
   string localFile,
   string remoteFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be created, overwritten or appended to. The file pathing and name conventions must be that of the local host.

remoteFile

A string that specifies the file on the server that will be transferred to the local system. The file pathing and name conventions must be that of the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncGetFile** method will download the contents of a remote file to a file on the local system. It is similar to the **GetFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being downloaded from the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create a new instance of this class for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is downloaded, the class will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to download more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.AsyncGetFile Overload List | AsyncPutFile Method

FtpClient.AsyncGetFile Method (String, String, FtpTransferOptions)

Download a file from the server to the local system in the background.

[Visual Basic]

```
Overloads Public Function AsyncGetFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal options As FtpTransferOptions _
) As Boolean
```

[C#]

```
public bool AsyncGetFile(
   string localFile,
   string remoteFile,
   FtpTransferOptions options
);
```

Parameters

localFile
A string that specifies the file on the local system that will be created, overwritten or appended to. The file pathing and name conventions must be that of the local host.

remoteFile
A string that specifies the file on the server that will be transferred to the local system. The file pathing and name conventions must be that of the server.

options
An FtpTransferOptions enumeration value which specifies one or more file transfer options.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncGetFile** method will download the contents of a remote file to a file on the local system. It is similar to the **GetFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being downloaded from the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create a new instance of this class for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is downloaded, the class will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to download more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.AsyncGetFile Overload List | AsyncPutFile Method

FtpClient.AsyncGetFile Method (String, String, FtpTransferOptions, Int64)

Download a file from the server to the local system in the background.

[Visual Basic]

```
Overloads Public Function AsyncGetFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal options As FtpTransferOptions, _
   ByVal offset As Long _
) As Boolean
```

[C#]

```
public bool AsyncGetFile(
   string localFile,
   string remoteFile,
   FtpTransferOptions options,
   long offset
);
```

Parameters

localFile

A string that specifies the file on the local system that will be created, overwritten or appended to. The file pathing and name conventions must be that of the local host.

remoteFile

A string that specifies the file on the server that will be transferred to the local system. The file pathing and name conventions must be that of the server.

options

An FtpTransferOptions enumeration value which specifies one or more file transfer options.

offset

A byte offset which specifies where the file transfer should begin. The default value of zero specifies that the file transfer should start at the beginning of the file. A value greater than zero is typically used to restart a transfer that has not completed successfully. Note that specifying a non-zero offset requires that the server support the REST command to restart transfers.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncGetFile** method will download the contents of a remote file to a file on the local system. It is similar to the **GetFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being downloaded from the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create a new instance of this class for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is downloaded, the class will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to download more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.AsyncGetFile Overload List | AsyncPutFile Method

FtpClient.AsyncPutFile Method

Upload a file from the local system to the server in the background.

Overload List

Upload a file from the local system to the server in the background.

public bool AsyncPutFile(string);

Upload a file from the local system to the server in the background.

public bool AsyncPutFile(string,string);

Upload a file from the local system to the server in the background.

public bool AsyncPutFile(string,string,FtpTransferOptions);

Upload a file from the local system to the server in the background.

public bool AsyncPutFile(string,string,FtpTransferOptions,long);

See Also

FtpClient Class | SocketTools Namespace | AsyncGetFile Method

FtpClient.AsyncPutFile Method (String)

Upload a file from the local system to the server in the background.

[Visual Basic]

```
Overloads Public Function AsyncPutFile( _
   ByVal localFile As String _
) As Boolean
```

[C#]

```
public bool AsyncPutFile(
   string localFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncPutFile** method will upload the contents of a file on the local system to the server. It is similar to the **PutFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being uploaded to the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create an instance of the control for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is uploaded, the control will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to upload more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.AsyncPutFile Overload List | AsyncGetFile Method

FtpClient.AsyncPutFile Method (String, String)

Upload a file from the local system to the server in the background.

[Visual Basic]

```
Overloads Public Function AsyncPutFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String _
) As Boolean
```

[C#]

```
public bool AsyncPutFile(
   string localFile,
   string remoteFile
);
```

Parameters

localFile
: A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

remoteFile
: A string that specifies the file on the server that will be created, overwritten or appended to. The file pathing and name conventions must be that of the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncPutFile** method will upload the contents of a file on the local system to the server. It is similar to the **PutFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being uploaded to the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create an instance of the control for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is uploaded, the control will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to upload more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.AsyncPutFile Overload List | AsyncGetFile Method

FtpClient.AsyncPutFile Method (String, String, FtpTransferOptions)

Upload a file from the local system to the server in the background.

[Visual Basic]

```
Overloads Public Function AsyncPutFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal options As FtpTransferOptions _
) As Boolean
```

[C#]

```
public bool AsyncPutFile(
   string localFile,
   string remoteFile,
   FtpTransferOptions options
);
```

Parameters

localFile
A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

remoteFile
A string that specifies the file on the server that will be created, overwritten or appended to. The file pathing and name conventions must be that of the server.

options
An FtpTransferOptions enumeration value which specifies one or more file transfer options.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncPutFile** method will upload the contents of a file on the local system to the server. It is similar to the **PutFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being uploaded to the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create an instance of the control for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is uploaded, the control will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to upload more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.AsyncPutFile Overload List | AsyncGetFile Method

FtpClient.AsyncPutFile Method (String, String, FtpTransferOptions, Int64)

Upload a file from the local system to the server in the background.

[Visual Basic]

```
Overloads Public Function AsyncPutFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal options As FtpTransferOptions, _
   ByVal offset As Long _
) As Boolean
```

[C#]

```
public bool AsyncPutFile(
   string localFile,
   string remoteFile,
   FtpTransferOptions options,
   long offset
);
```

Parameters

localFile

A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

remoteFile

A string that specifies the file on the server that will be created, overwritten or appended to. The file pathing and name conventions must be that of the server.

options

An FtpTransferOptions enumeration value which specifies one or more file transfer options.

offset

A byte offset which specifies where the file transfer should begin. The default value of zero specifies that the file transfer should start at the beginning of the file. A value greater than zero is typically used to restart a transfer that has not completed successfully. Note that specifying a non-zero offset requires that the server support the REST command to restart transfers.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncPutFile** method will upload the contents of a file on the local system to the server. It is similar to the **PutFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being uploaded to the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create an instance of the control for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is uploaded, the control will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to upload more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.AsyncPutFile Overload List | AsyncGetFile Method

FtpClient.Cancel Method

Cancel the current blocking client operation.

[Visual Basic]

```
Public Sub Cancel()
```

[C#]

```
public void Cancel();
```

Remarks

When the **Cancel** method is called, the blocking client operation will not immediately fail. An internal flag is set which causes the blocking operation to exit with an error. This means that the application cannot cancel an operation and immediately perform some other blocking function. Instead it must allow the calling stack to unwind, returning back to the blocking operation before making any further function calls.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.ChangeDirectory Method

Change the current working directory on the remote server.

[Visual Basic]

```
Public Function ChangeDirectory( _
   ByVal pathName As String _
) As Boolean
```

[C#]

```
public bool ChangeDirectory(
   string pathName
);
```

Parameters

pathName
A string which specifies the directory on the remote server. The file pathing and name conventions must be that of the remote host.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.CloseDirectory Method

Close the directory that was previously opened with the OpenDirectory method.

[Visual Basic]

```
Public Function CloseDirectory() As Boolean
```

[C#]

```
public bool CloseDirectory();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.CloseFile Method

Close the file that was previously opened with the OpenFile method.

[Visual Basic]

```
Public Function CloseFile() As Boolean
```

[C#]

```
public bool CloseFile();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Command Method

Send a custom command to the server.

Overload List

Send a custom command to the server.

public bool Command(string);

Send a custom command to the server.

public bool Command(string,string);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Command Method (String)

Send a custom command to the server.

[Visual Basic]

```
Overloads Public Function Command( _
   ByVal command As String _
) As Boolean
```

[C#]

```
public bool Command(
   string command
);
```

Parameters

command

A string which specifies the command to send. Valid commands vary based on the Internet protocol and the type of server that the client is connected to. Consult the protocol standard and/or the technical reference documentation for the server to determine what commands may be issued by a client application.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Command** method sends a command to the remote host and processes the result code sent back in response to that command. This method can be used to send custom commands to a server to take advantage of features or capabilities that may not be supported internally by the class library.

To determine the specific status code returned by the server, check the value of the **ResultCode** property after the method returns.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Command Overload List

FtpClient.Command Method (String, String)

Send a custom command to the server.

[Visual Basic]

```
Overloads Public Function Command( _
   ByVal command As String, _
   ByVal parameters As String _
) As Boolean
```

[C#]

```
public bool Command(
   string command,
   string parameters
);
```

Parameters

command

A string which specifies the command to send. Valid commands vary based on the Internet protocol and the type of server that the client is connected to. Consult the protocol standard and/or the technical reference documentation for the server to determine what commands may be issued by a client application.

parameters

An string which specifies one or more parameters to be sent along with the command. If more than one parameter is required, they must be separated by a single space character. Consult the protocol standard and/or technical reference documentation for the server to determine what parameters should be provided when issuing a specific command.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Command** method sends a command to the remote host and processes the result code sent back in response to that command. This method can be used to send custom commands to a server to take advantage of features or capabilities that may not be supported internally by the class library.

To determine the specific status code returned by the server, check the value of the **ResultCode** property after the method returns.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Command Overload List

FtpClient.Connect Method

Establish a connection with a remote host.

Overload List

Establish a connection with a remote host.

public bool Connect();

Establish a connection with a remote host.

public bool Connect(string);

Establish a connection with a remote host.

public bool Connect(string,int);

Establish a connection with a remote host.

public bool Connect(string,int,string,string);

Establish a connection with a remote host.

public bool Connect(string,int,string,string,int);

Establish a connection with a remote host.

public bool Connect(string,int,string,string,int,FtpOptions);

Establish a connection with a remote host.

public bool Connect(string,int,string,string,string);

Establish a connection with a remote host.

public bool Connect(string,int,string,string,string,int);

Establish a connection with a remote host.

public bool Connect(string,int,string,string,string,int,FtpOptions);

Establish a connection with a remote host.

public bool Connect(string,string,string);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Connect Method ()

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect() As Boolean
```

[C#]

```
public bool Connect();
```

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Connect Overload List

FtpClient.Connect Method (String)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Connect Overload List

FtpClient.Connect Method (String, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Connect Overload List

FtpClient.Connect Method (String, Int32, String, String)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName

A string which specifies the user name which will be used to authenticate the client session. If the user name is specified as an empty string, then the login is considered to be anonymous.

userPassword

A string which specifies the password which will be used to authenticate the client session. This argument may be an empty string if no password is required for the specified user, or if no username has been specified.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Connect Overload List

FtpClient.Connect Method (String, Int32, String, String, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   int timeout
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName

A string which specifies the user name which will be used to authenticate the client session. If the user name is specified as an empty string, then the login is considered to be anonymous.

userPassword

A string which specifies the password which will be used to authenticate the client session. This argument may be an empty string if no password is required for the specified user, or if no username has been specified.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Connect Overload List

FtpClient.Connect Method (String, Int32, String, String, Int32, FtpOptions)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal timeout As Integer, _
   ByVal options As FtpOptions _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   int timeout,
   FtpOptions options
);
```

Parameters

hostName
A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort
An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName
A string which specifies the user name which will be used to authenticate the client session. If the user name is specified as an empty string, then the login is considered to be anonymous.

userPassword
A string which specifies the password which will be used to authenticate the client session. This argument may be an empty string if no password is required for the specified user, or if no username has been specified.

timeout
An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

options
One or more of the FtpOptions enumeration flags.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Connect Overload List

FtpClient.Connect Method (String, Int32, String, String, String)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal userAccount As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   string userAccount
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName

A string which specifies the user name which will be used to authenticate the client session. If the user name is specified as an empty string, then the login is considered to be anonymous.

userPassword

A string which specifies the password which will be used to authenticate the client session. This argument may be an empty string if no password is required for the specified user, or if no username has been specified.

userAccount

A string that specifies the account name to be used to authenticate the current client session. This parameter may be an empty string if no account name is required for the specified user.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Connect Overload List

FtpClient.Connect Method (String, Int32, String, String, String, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal userAccount As String, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   string userAccount,
   int timeout
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName

A string which specifies the user name which will be used to authenticate the client session. If the user name is specified as an empty string, then the login is considered to be anonymous.

userPassword

A string which specifies the password which will be used to authenticate the client session. This argument may be an empty string if no password is required for the specified user, or if no username has been specified.

userAccount

A string that specifies the account name to be used to authenticate the current client session. This parameter may be an empty string if no account name is required for the specified user.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Connect Overload List

FtpClient.Connect Method (String, Int32, String, String, String, Int32, FtpOptions)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal userAccount As String, _
   ByVal timeout As Integer, _
   ByVal options As FtpOptions _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   string userAccount,
   int timeout,
   FtpOptions options
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName

A string which specifies the user name which will be used to authenticate the client session. If the user name is specified as an empty string, then the login is considered to be anonymous.

userPassword

A string which specifies the password which will be used to authenticate the client session. This argument may be an empty string if no password is required for the specified user, or if no username has been specified.

userAccount

A string that specifies the account name to be used to authenticate the current client session. This parameter may be an empty string if no account name is required for the specified user.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

options

One or more of the FtpOptions enumeration flags.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Connect Overload List

FtpClient.Connect Method (String, String, String)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   string userName,
   string userPassword
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

userName

A string which specifies the user name which will be used to authenticate the client session. If the user name is specified as an empty string, then the login is considered to be anonymous.

userPassword

A string which specifies the password which will be used to authenticate the client session. This argument may be an empty string if no password is required for the specified user, or if no username has been specified.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Connect Overload List

FtpClient.CreateFile Method

Create a new file or overwrite an existing file.

[Visual Basic]

```
Public Function CreateFile( _
   ByVal remoteFile As String _
) As Boolean
```

[C#]

```
public bool CreateFile(
   string remoteFile
);
```

Parameters

remoteFile

A string which specifies the name of the file to create on the remote server. The file pathing and name conventions must be that of the remote host.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateFile** method creates a new file on the remote server using the specified file name. The **Write** method may then be used to copy data to the open file. The user must have the appropriate permission to create the file on the server

See Also

FtpClient Class | SocketTools Namespace

FtpClient.DeleteFile Method

Delete a file on the remote server.

[Visual Basic]

```
Public Function DeleteFile( _
   ByVal remoteFile As String _
) As Boolean
```

[C#]

```
public bool DeleteFile(
   string remoteFile
);
```

Parameters

remoteFile

A string which specifies the name of the file on the remote server that is to be deleted. The file pathing and name conventions must be that of the remote host.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **DeleteFile** method deletes an existing file from the remote server. The user must have the appropriate permission to delete the specified file.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Disconnect Method

Terminate the connection with the remote server.

[Visual Basic]

```
Public Sub Disconnect()
```

[C#]

```
public void Disconnect();
```

Remarks

The **Disconnect** method terminates the connection with the remote host and closes the socket handle allocated by the class. Note that the socket is not immediately released when the connection is terminated and will enter a wait state for two minutes. After the time wait period has elapsed, the socket will be released by the operating system. This is a normal safety mechanism to handle any packets that may arrive after the connection has been closed.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Dispose Method

Releases all resources used by FtpClient.

Overload List

Releases all resources used by FtpClient.

public void Dispose();

Releases the unmanaged resources allocated by the FtpClient class and optionally releases the managed resources.

protected virtual void Dispose(bool);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the FtpClient class and optionally releases the managed resources.

[Visual Basic]

```
Overridable Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]

```
protected virtual void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **FtpClient** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Dispose Overload List

FtpClient.Dispose Method ()

Releases all resources used by FtpClient.

[Visual Basic]

```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]

```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Dispose Overload List

FtpClient.Finalize Method

Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.

[Visual Basic]

```
Overrides Protected Sub Finalize()
```

[C#]

```
protected override void Finalize();
```

See Also

FtpClient Class | SocketTools Namespace

FtpClient.GetData Method

Transfers the contents of a file on the server and stores it in a byte array.

Overload List

Transfers the contents of a file on the server and stores it in a byte array.

public bool GetData(string,byte[],ref int);

Transfers the contents of a file from the server and stores it in a MemoryStream.

public bool GetData(string,MemoryStream);

Transfers the contents of a file on the server and stores it in a string.

public bool GetData(string,ref string);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.GetData Method (String, Byte[], Int32)

Transfers the contents of a file on the server and stores it in a byte array.

[Visual Basic]

```
Overloads Public Function GetData( _
   ByVal remoteFile As String, _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool GetData(
   string remoteFile,
   byte[] buffer,
   ref int length
);
```

Parameters

remoteFile

A string that specifies the file on the remote system that will be transferred to the local system. The file pathing and name conventions must be that of the remote host.

buffer

A byte array that the data will be stored in.

length

An integer value passed by reference which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller. When the method returns, this value will be updated with the number of bytes actually stored in the byte array.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetData** method transfers data from a file on the server to the local system, storing it in the specified buffer . This method will cause the calling current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetData Overload List

FtpClient.GetData Method (String, String)

Transfers the contents of a file on the server and stores it in a string.

[Visual Basic]

```
Overloads Public Function GetData( _
   ByVal remoteFile As String, _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool GetData(
   string remoteFile,
   ref string buffer
);
```

Parameters

remoteFile

A string that specifies the file on the remote system that will be transferred to the local system. The file pathing and name conventions must be that of the remote host.

buffer

A string passed by reference that the data will be stored in. It is not recommended that binary files be stored in a string buffer. Only text files should be downloaded using this implementation of the method. For binary files, store the data in a byte array instead.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetData** method transfers data from a file on the server to the local system, storing it in the specified buffer . This method will cause the calling current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetData Overload List

FtpClient.GetData Method (String, MemoryStream)

Transfers the contents of a file from the server and stores it in a MemoryStream.

[Visual Basic]

```
Overloads Public Function GetData( _
   ByVal remoteFile As String, _
   ByVal memStream As MemoryStream _
) As Boolean
```

[C#]

```
public bool GetData(
   string remoteFile,
   MemoryStream memStream
);
```

Parameters

remoteFile

A string that specifies the file on the remote system that will be transferred to the local system. The file pathing and name conventions must be that of the remote host.

memStream

A System.IO.MemoryStream object that will contain the file data when the method returns. This stream must be open and writable.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetData** method transfers data from a file on the server to the local system, storing it in the specified MemoryStream. This method will cause the calling current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The contents of the MemoryStream will be replaced by the contents of the file and the current position will be reset to the beginning of the stream. The stream must be open and writable, otherwise this method will throw **System.NotSupportedException**.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetData Overload List

FtpClient.GetDirectory Method

Return the current working directory.

[Visual Basic]

```
Public Function GetDirectory( _
   ByRef pathName As String _
) As Boolean
```

[C#]

```
public bool GetDirectory(
   ref string pathName
);
```

Parameters

pathName

A string passed by reference which will contain the current working directory on the server when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.GetFile Method

Download a file from the server to the local system.

Overload List

Download a file from the server to the local system.

public bool GetFile(string);

Download a file from the server to the local system.

public bool GetFile(string,string);

Download a file from the server to the local system.

public bool GetFile(string,string,FtpTransferOptions);

Download a file from the server to the local system.

public bool GetFile(string,string,FtpTransferOptions,long);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.GetFile Method (String)

Download a file from the server to the local system.

[Visual Basic]

```
Overloads Public Function GetFile( _
   ByVal localFile As String _
) As Boolean
```

[C#]

```
public bool GetFile(
   string localFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be created, overwritten or appended to. The file pathing and name conventions must be that of the local host.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFile** method copies an existing file from the server to the local system. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

This implementation of the **GetFile** method is most commonly used when the name of the remote file has already been specified, either by setting the **URL** property or by explicitly setting the **RemoteFile** property.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetFile Overload List

FtpClient.GetFile Method (String, String)

Download a file from the server to the local system.

[Visual Basic]

```
Overloads Public Function GetFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String _
) As Boolean
```

[C#]

```
public bool GetFile(
   string localFile,
   string remoteFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be created, overwritten or appended to. The file pathing and name conventions must be that of the local host.

remoteFile

A string that specifies the file on the remote system that will be transferred to the local system. The file pathing and name conventions must be that of the remote host.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFile** method copies an existing file from the server to the local system. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetFile Overload List

FtpClient.GetFile Method (String, String, FtpTransferOptions)

Download a file from the server to the local system.

[Visual Basic]

```
Overloads Public Function GetFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal options As FtpTransferOptions _
) As Boolean
```

[C#]

```
public bool GetFile(
   string localFile,
   string remoteFile,
   FtpTransferOptions options
);
```

Parameters

localFile
A string that specifies the file on the local system that will be created, overwritten or appended to. The file pathing and name conventions must be that of the local host.

remoteFile
A string that specifies the file on the remote system that will be transferred to the local system. The file pathing and name conventions must be that of the remote host.

options
An FtpTransferOptions enumeration value which specifies one or more file transfer options.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFile** method copies an existing file from the server to the local system. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetFile Overload List

FtpClient.GetFile Method (String, String, FtpTransferOptions, Int64)

Download a file from the server to the local system.

[Visual Basic]

```
Overloads Public Function GetFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal options As FtpTransferOptions, _
   ByVal offset As Long _
) As Boolean
```

[C#]

```
public bool GetFile(
   string localFile,
   string remoteFile,
   FtpTransferOptions options,
   long offset
);
```

Parameters

localFile
: A string that specifies the file on the local system that will be created, overwritten or appended to. The file pathing and name conventions must be that of the local host.

remoteFile
: A string that specifies the file on the remote system that will be transferred to the local system. The file pathing and name conventions must be that of the remote host.

options
: An FtpTransferOptions enumeration value which specifies one or more file transfer options.

offset
: A integer value which specifies the offset where the file transfer should begin. A value of zero specifies that the file transfer should start at the beginning of the file. A value greater than zero is typically used to restart a transfer that has not completed successfully. Note that specifying a non-zero offset requires that the server support the REST command to restart file transfers.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFile** method copies an existing file from the server to the local system. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetFile Overload List

FtpClient.GetFileList Method

Returns an unparsed list of files in the specified directory.

Overload List

Returns an unparsed list of files in the specified directory.

public bool GetFileList(string,ref string);

Returns an unparsed list of files in the specified directory.

public bool GetFileList(string,ref string,bool);

Returns an unparsed list of files in the current working directory.

public bool GetFileList(ref string);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.GetFileList Method (String)

Returns an unparsed list of files in the current working directory.

[Visual Basic]

```
Overloads Public Function GetFileList( _
   ByRef fileList As String _
) As Boolean
```

[C#]

```
public bool GetFileList(
   ref string fileList
);
```

Parameters

fileList

A string buffer passed by reference that will contain the file list data when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFileList** method returns a list of files in the current working directory on the server, copying the data to a string buffer. Unlike the **GetFirstFile** and **GetNextFile** methods that parses a directory listing, this method returns the unparsed file list data. The actual format of the data that is returned depends on the operating system and how the server implements file listings. For example, UNIX servers typically return the output from the **/bin/ls** command.

This method can be particularly useful when the client is connected to a server that returns file listings in a format that is not recognized by the component. The application can retrieve the unparsed file listing from the server and parse the contents.

This method is supported for both FTP and SFTP (SSH) connections, however the format of the data may differ depending on which protocol is used. Most UNIX based FTP servers will not list files and subdirectories that begin with a period, however most SFTP servers will return a list of all files, even those that begin with a period.

This method will cause the current thread to block until the file listing completes, a timeout occurs or the operation is canceled.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetFileList Overload List

FtpClient.GetFileList Method (String, String)

Returns an unparsed list of files in the specified directory.

[Visual Basic]

```
Overloads Public Function GetFileList( _
   ByVal remotePath As String, _
   ByRef fileList As String _
) As Boolean
```

[C#]

```
public bool GetFileList(
   string remotePath,
   ref string fileList
);
```

Parameters

remotePath

A string value which specifies the name of a directory on the server. The list of files and subdirectories in that directory will be returned to the client. To obtain a list of files in the current working directory on the server, use an empty string.

fileList

A string buffer passed by reference that will contain the file list data when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFileList** method returns a list of files in the specified directory, copying the data to a string buffer. Unlike the **GetFirstFile** and **GetNextFile** methods that parses a directory listing, this method returns the unparsed file list data. The actual format of the data that is returned depends on the operating system and how the server implements file listings. For example, UNIX servers typically return the output from the **/bin/ls** command.

Some servers may not support file listings for any directory other than the current working directory. If an error is returned when specifying a directory name, try changing the current working directory using the **ChangeDirectory** method and then call this method again, an empty string as the ***remotePath*** parameter.

This method can be particularly useful when the client is connected to a server that returns file listings in a format that is not recognized by the component. The application can retrieve the unparsed file listing from the server and parse the contents.

This method is supported for both FTP and SFTP (SSH) connections, however the format of the data may differ depending on which protocol is used. Most UNIX based FTP servers will not list files and subdirectories that begin with a period, however most SFTP servers will return a list of all files, even those that begin with a period.

This method will cause the current thread to block until the file listing completes, a timeout occurs or the operation is canceled.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetFileList Overload List

FtpClient.GetFileList Method (String, String, Boolean)

Returns an unparsed list of files in the specified directory.

[Visual Basic]

```
Overloads Public Function GetFileList( _
   ByVal remotePath As String, _
   ByRef fileList As String, _
   ByVal nameOnly As Boolean _
) As Boolean
```

[C#]

```
public bool GetFileList(
   string remotePath,
   ref string fileList,
   bool nameOnly
);
```

Parameters

remotePath

A string value which specifies the name of a directory on the server. The list of files and subdirectories in that directory will be returned to the client. To obtain a list of files in the current working directory on the server, use an empty string.

fileList

A string buffer passed by reference that will contain the file list data when the method returns.

nameOnly

A boolean value that specifies if the file listing should contain only the file name, or should include additional information about the file such as the size, ownership and permissions.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFileList** method returns a list of files in the specified directory, copying the data to a string buffer. Unlike the **GetFirstFile** and **GetNextFile** methods that parses a directory listing, this method returns the unparsed file list data. The actual format of the data that is returned depends on the operating system and how the server implements file listings. For example, UNIX servers typically return the output from the **/bin/ls** command.

Some servers may not support file listings for any directory other than the current working directory. If an error is returned when specifying a directory name, try changing the current working directory using the **ChangeDirectory** method and then call this method again, an empty string as the *remotePath* parameter.

This method can be particularly useful when the client is connected to a server that returns file listings in a format that is not recognized by the component. The application can retrieve the unparsed file listing from the server and parse the contents. Note that if you specify the *nameOnly* parameter as true, the data will only contain a list of file names and there will be no way for the application to know if they represent a regular file or a subdirectory.

This method is supported for both FTP and SFTP (SSH) connections, however the format of the data may differ depending on which protocol is used. Most UNIX based FTP servers will not list files and subdirectories that begin with a period, however most SFTP servers will return a list of all files, even those that begin with a period.

This method will cause the current thread to block until the file listing completes, a timeout occurs or the operation is canceled.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetFileList Overload List

FtpClient.GetFilePermissions Method

Return the access permissions for a file on the remote system.

[Visual Basic]

```
Public Function GetFilePermissions( _
   ByVal remoteFile As String, _
   ByRef filePerms As FtpPermissions _
) As Boolean
```

[C#]

```
public bool GetFilePermissions(
   string remoteFile,
   ref FtpPermissions filePerms
);
```

Parameters

remoteFile

A string that specifies the name of the file that the access permissions are to be returned for. The filename cannot contain any wildcard characters.

filePerms

An FtpPermissions enumeration value which is passed by reference and set to the file permissions when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFilePermissions** method returns information about the access permissions for a specific file on the server. This method uses the STAT command to retrieve information about the specified file. If the server does not support the use of this command, an error will be returned. You can use the **Features** property to determine what features are available and/or enabled on the server.

Note that on some systems, the STAT command will not return information on files that contain spaces or tabs in the filename. In this case, the method will fail.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.GetFileSize Method

Returns the size of the specified file on the remote server.

Overload List

Returns the size of the specified file on the remote server.

public bool GetFileSize(string,ref int);

Returns the size of the specified file on the remote server.

public bool GetFileSize(string,ref long);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.GetFileSize Method (String, Int64)

Returns the size of the specified file on the remote server.

[Visual Basic]

```
Overloads Public Function GetFileSize( _
   ByVal remoteFile As String, _
   ByRef fileSize As Long _
) As Boolean
```

[C#]

```
public bool GetFileSize(
   string remoteFile,
   ref long fileSize
);
```

Parameters

remoteFile

A string that specifies the name of the file on the server. The filename cannot contain any wildcard characters and must follow the naming conventions of the operating system the server is hosted on.

fileSize

A long integer value which is passed by reference and will specify the size of the file on the server when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method uses the SIZE command to determine the length of the specified file. Not all servers implement this command, in which case the method will fail. You can use the **Features** property to determine what features are available and/or enabled on the server.

Note that if the file on the server is a text file, it is possible that the value returned by this method will not match the size of the file when it is downloaded to the local system. This is because different operating systems use different sequences of characters to mark the end of a line of text, and when a file is transferred in text mode, the end of line character sequence is automatically converted to a carriage return-linefeed, which is the convention used by the Windows platform.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetFileSize Overload List

FtpClient.GetFileSize Method (String, Int32)

Returns the size of the specified file on the remote server.

[Visual Basic]

```
Overloads Public Function GetFileSize( _
   ByVal remoteFile As String, _
   ByRef fileSize As Integer _
) As Boolean
```

[C#]

```
public bool GetFileSize(
   string remoteFile,
   ref int fileSize
);
```

Parameters

remoteFile

A string that specifies the name of the file on the server. The filename cannot contain any wildcard characters and must follow the naming conventions of the operating system the server is hosted on.

fileSize

A long integer value which is passed by reference and will specify the size of the file on the server when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method uses the SIZE command to determine the length of the specified file. Not all servers implement this command, in which case the method will fail. You can use the **Features** property to determine what features are available and/or enabled on the server.

Note that if the file on the server is a text file, it is possible that the value returned by this method will not match the size of the file when it is downloaded to the local system. This is because different operating systems use different sequences of characters to mark the end of a line of text, and when a file is transferred in text mode, the end of line character sequence is automatically converted to a carriage return-linefeed, which is the convention used by the Windows platform.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetFileSize Overload List

FtpClient.GetFileStatus Method

Returns status information about the specified file.

[Visual Basic]

```
Public Function GetFileStatus( _
   ByVal remoteFile As String, _
   ByRef fileInfo As FileInformation _
) As Boolean
```

[C#]

```
public bool GetFileStatus(
   string remoteFile,
   ref FileInformation fileInfo
);
```

Parameters

remoteFile

A string which specifies the name of the file that status information is to be returned for.

fileInfo

A FileInformation structure that is passed by reference. When the method returns, the members of this structure will be populated with information about the file on the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFileStatus** method returns information about the specified file. The filename must be specified using the remote host file naming conventions, and cannot include wildcard characters. The primary difference between using this method and using the **OpenDirectory, GetFirstFile** and **GetNextFile** methods to obtain file information is that the file status information is returned on the command channel. This method cannot be used while a file transfer is in progress or while a file listing is being returned by the server.

This method requires that the server return file status information in response to the STAT command. Some servers, for example on VMS platforms, do not provide this information. On some systems, the STAT command will not return information on files that contain spaces or tabs in the filename. In this case, the method will return an empty structure.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.GetFileTime Method

Returns the modification date and time for specified file on the remote server.

Overload List

Returns the modification date and time for specified file on the remote server.

public bool GetFileTime(string,ref DateTime);

Returns the modification date and time for specified file on the remote server.

public bool GetFileTime(string,ref DateTime,bool);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.GetFileTime Method (String, DateTime)

Returns the modification date and time for specified file on the remote server.

[Visual Basic]

```
Overloads Public Function GetFileTime( _
   ByVal remoteFile As String, _
   ByRef fileDate As Date _
) As Boolean
```

[C#]

```
public bool GetFileTime(
   string remoteFile,
   ref DateTime fileDate
);
```

Parameters

remoteFile

A string that specifies the name of the file on the server. The filename cannot contain any wildcard characters and must follow the naming conventions of the operating system the server is hosted on.

fileDate

A **System.DateTime** structure which is passed by reference. When the method returns, this object will be set to the date and time that the file was created or last modified.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method uses the MTDM command to determine the modification time for the file. If the server does not support this command, the method will attempt to use the STAT command to determine the file modification time. You can use the **Features** property to determine what features are available and/or enabled on the server.

The value of the Localize property determines if the date and time are adjusted for the local timezone.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetFileTime Overload List

FtpClient.GetFileTime Method (String, DateTime, Boolean)

Returns the modification date and time for specified file on the remote server.

[Visual Basic]

```
Overloads Public Function GetFileTime( _
   ByVal remoteFile As String, _
   ByRef fileDate As Date, _
   ByVal localDate As Boolean _
) As Boolean
```

[C#]

```
public bool GetFileTime(
   string remoteFile,
   ref DateTime fileDate,
   bool localDate
);
```

Parameters

remoteFile

A string that specifies the name of the file on the server. The filename cannot contain any wildcard characters and must follow the naming conventions of the operating system the server is hosted on.

fileDate

A **System.DateTime** structure which is passed by reference. When the method returns, this object will be set to the date and time that the file was created or last modified.

localDate

A boolean flag which specifies if the date and time for the file should adjusted for the local timezone. A value of **true** specifies that the date and time should be adjusted for the local timezone. A value of **false** specifies that the date and time should be returned as a UTC (Coordinated Universal Time) value.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method uses the MTDM command to determine the modification time for the file. If the server does not support this command, the method will attempt to use the STAT command to determine the file modification time. You can use the **Features** property to determine what features are available and/or enabled on the server.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetFileTime Overload List

FtpClient.GetFirstFile Method

Get information about the first file in a directory listing returned by the server.

Overload List

Get information about the first file in a directory listing returned by the server.

public bool GetFirstFile(ref FileInformation);

Get the first file name in a directory listing returned by the server.

public bool GetFirstFile(ref string);

Get information about the first file in a directory listing returned by the server.

public bool GetFirstFile(ref string,ref bool);

Get information about the first file in a directory listing returned by the server.

public bool GetFirstFile(ref string,ref long,ref bool);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.GetFirstFile Method (FileInformation)

Get information about the first file in a directory listing returned by the server.

[Visual Basic]

```
Overloads Public Function GetFirstFile( _
   ByRef fileInfo As FileInformation _
) As Boolean
```

[C#]

```
public bool GetFirstFile(
   ref FileInformation fileInfo
);
```

Parameters

fileInfo

A FileInformation structure that is passed by reference. When the method returns, the members of this structure will be populated with information about the file on the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

A file listing is initiated by calling the **OpenDirectory** method. Then, the application must call **GetFirstFile**, followed by calling **GetNextFile** in a loop until the method returns **false**. Once the complete directory listing has been returned, the **CloseDirectory** method must be called.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetFirstFile Overload List

FtpClient.GetFirstFile Method (String)

Get the first file name in a directory listing returned by the server.

[Visual Basic]

```
Overloads Public Function GetFirstFile( _
   ByRef fileName As String _
) As Boolean
```

[C#]

```
public bool GetFirstFile(
   ref string fileName
);
```

Parameters

fileName

A string passed by reference which will contain a file name when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

A file listing is initiated by calling the **OpenDirectory** method. Then, the application must call **GetFirstFile**, followed by calling **GetNextFile** in a loop until the method returns **false**. Once the complete directory listing has been returned, the **CloseDirectory** method must be called.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetFirstFile Overload List

FtpClient.GetFirstFile Method (String, Boolean)

Get information about the first file in a directory listing returned by the server.

[Visual Basic]

```
Overloads Public Function GetFirstFile( _
   ByRef fileName As String, _
   ByRef isDirectory As Boolean _
) As Boolean
```

[C#]

```
public bool GetFirstFile(
   ref string fileName,
   ref bool isDirectory
);
```

Parameters

fileName
A string passed by reference which will contain a file name when the method returns.

isDirectory
A boolean passed by reference which will specify if the file is a regular file or a directory. A value of **true** indicates that the file is a directory. A value of **false** indicates that it is a regular file.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

A file listing is initiated by calling the **OpenDirectory** method. Then, the application must call **GetFirstFile**, followed by calling **GetNextFile** in a loop until the method returns **false**. Once the complete directory listing has been returned, the **CloseDirectory** method must be called.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetFirstFile Overload List

FtpClient.GetFirstFile Method (String, Int64, Boolean)

Get information about the first file in a directory listing returned by the server.

[Visual Basic]

```
Overloads Public Function GetFirstFile( _
   ByRef fileName As String, _
   ByRef fileSize As Long, _
   ByRef isDirectory As Boolean _
) As Boolean
```

[C#]

```
public bool GetFirstFile(
   ref string fileName,
   ref long fileSize,
   ref bool isDirectory
);
```

Parameters

fileName
A string passed by reference which will contain a file name when the method returns.

fileSize
An integer passed by reference which will contain the size of the file when the method returns.

isDirectory
A boolean passed by reference which will specify if the file is a regular file or a directory. A value of **true** indicates that the file is a directory. A value of **false** indicates that it is a regular file.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

A file listing is initiated by calling the **OpenDirectory** method. Then, the application must call **GetFirstFile**, followed by calling **GetNextFile** in a loop until the method returns **false**. Once the complete directory listing has been returned, the **CloseDirectory** method must be called.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetFirstFile Overload List

FtpClient.GetMultipleFiles Method

Download multiple files from the server to the local system using a wildcard mask.

Overload List

Download multiple files from the server to the local system using a wildcard mask.

public bool GetMultipleFiles(string,string);

Download multiple files from the server to the local system using a wildcard mask.

public bool GetMultipleFiles(string,string,string);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.GetMultipleFiles Method (String, String, String)

Download multiple files from the server to the local system using a wildcard mask.

[Visual Basic]

```
Overloads Public Function GetMultipleFiles( _
   ByVal localPath As String, _
   ByVal remotePath As String, _
   ByVal fileMask As String _
) As Boolean
```

[C#]

```
public bool GetMultipleFiles(
   string localPath,
   string remotePath,
   string fileMask
);
```

Parameters

localPath

A string argument which specifies the name of the directory on the local system where the files will be stored. If a file by the same name already exists, it will be overwritten

remotePath

A string argument which specifies the name of the directory on the remote system where the files will be copied from. You must have permission to read the contents of the directory.

fileMask

An string argument which specifies the wildcard mask to be used when selecting what files should be transferred. Typically, this argument is a wildcard mask that limits the files downloaded from the server to those which match a specific extension.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMultipleFiles** method copies multiple files from the remote system to the local system. If the local file already exists, it is overwritten. This method will cause the current thread to block until all of the files have been transferred, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetMultipleFiles Overload List

FtpClient.GetMultipleFiles Method (String, String)

Download multiple files from the server to the local system using a wildcard mask.

[Visual Basic]

```
Overloads Public Function GetMultipleFiles( _
   ByVal localPath As String, _
   ByVal remotePath As String _
) As Boolean
```

[C#]

```
public bool GetMultipleFiles(
   string localPath,
   string remotePath
);
```

Parameters

localPath

A string argument which specifies the name of the directory on the local system where the files will be stored. If a file by the same name already exists, it will be overwritten

remotePath

A string argument which specifies the name of the directory on the remote system where the files will be copied from. You must have permission to read the contents of the directory.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMultipleFiles** method copies multiple files from the remote system to the local system. If the local file already exists, it is overwritten. This method will cause the current thread to block until all of the files have been transferred, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetMultipleFiles Overload List

FtpClient.GetNextFile Method

Get information about the next file in a directory listing returned by the server.

Overload List

Get information about the next file in a directory listing returned by the server.

public bool GetNextFile(ref FileInformation);

Get the next file name in a directory listing returned by the server.

public bool GetNextFile(ref string);

Get information about the next file in a directory listing returned by the server.

public bool GetNextFile(ref string,ref bool);

Get information about the next file in a directory listing returned by the server.

public bool GetNextFile(ref string,ref long,ref bool);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.GetNextFile Method (FileInformation)

Get information about the next file in a directory listing returned by the server.

[Visual Basic]

```
Overloads Public Function GetNextFile( _
   ByRef fileInfo As FileInformation _
) As Boolean
```

[C#]

```
public bool GetNextFile(
   ref FileInformation fileInfo
);
```

Parameters

fileInfo

A FileInformation structure that is passed by reference. When the method returns, the members of this structure will be populated with information about the file on the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

A file listing is initiated by calling the **OpenDirectory** method. Then, the application must call **GetFirstFile**, followed by calling **GetNextFile** in a loop until the method returns **false**. Once the complete directory listing has been returned, the **CloseDirectory** method must be called.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetNextFile Overload List

FtpClient.GetNextFile Method (String)

Get the next file name in a directory listing returned by the server.

[Visual Basic]

```
Overloads Public Function GetNextFile( _
   ByRef fileName As String _
) As Boolean
```

[C#]

```
public bool GetNextFile(
   ref string fileName
);
```

Parameters

fileName

A string passed by reference which will contain a file name when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

A file listing is initiated by calling the **OpenDirectory** method. Then, the application must call **GetFirstFile**, followed by calling **GetNextFile** in a loop until the method returns **false**. Once the complete directory listing has been returned, the **CloseDirectory** method must be called.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetNextFile Overload List

FtpClient.GetNextFile Method (String, Boolean)

Get information about the next file in a directory listing returned by the server.

[Visual Basic]

```
Overloads Public Function GetNextFile( _
   ByRef fileName As String, _
   ByRef isDirectory As Boolean _
) As Boolean
```

[C#]

```
public bool GetNextFile(
   ref string fileName,
   ref bool isDirectory
);
```

Parameters

fileName

A string passed by reference which will contain a file name when the method returns.

isDirectory

A boolean passed by reference which will specify if the file is a regular file or a directory. A value of **true** indicates that the file is a directory. A value of **false** indicates that it is a regular file.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

A file listing is initiated by calling the **OpenDirectory** method. Then, the application must call **GetFirstFile**, followed by calling **GetNextFile** in a loop until the method returns **false**. Once the complete directory listing has been returned, the **CloseDirectory** method must be called.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetNextFile Overload List

FtpClient.GetNextFile Method (String, Int64, Boolean)

Get information about the next file in a directory listing returned by the server.

[Visual Basic]

```
Overloads Public Function GetNextFile( _
   ByRef fileName As String, _
   ByRef fileSize As Long, _
   ByRef isDirectory As Boolean _
) As Boolean
```

[C#]

```
public bool GetNextFile(
   ref string fileName,
   ref long fileSize,
   ref bool isDirectory
);
```

Parameters

fileName
A string passed by reference which will contain a file name when the method returns.

fileSize
An integer passed by reference which will contain the size of the file when the method returns.

isDirectory
A boolean passed by reference which will specify if the file is a regular file or a directory. A value of **true** indicates that the file is a directory. A value of **false** indicates that it is a regular file.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

A file listing is initiated by calling the **OpenDirectory** method. Then, the application must call **GetFirstFile**, followed by calling **GetNextFile** in a loop until the method returns **false**. Once the complete directory listing has been returned, the **CloseDirectory** method must be called.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.GetNextFile Overload List

FtpClient.Initialize Method

Initialize an instance of the FtpClient class.

Overload List

Initialize an instance of the FtpClient class.

public bool Initialize();

Initialize an instance of the FtpClient class.

public bool Initialize(string);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Initialize Method ()

Initialize an instance of the FtpClient class.

[Visual Basic]

```
Overloads Public Function Initialize() As Boolean
```

[C#]

```
public bool Initialize();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Initialize** method can be used to explicitly initialize an instance of the **FtpClient** class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the **FtpClient** class

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Initialize Overload List

FtpClient.Initialize Method (String)

Initialize an instance of the FtpClient class.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Initialize** method can be used to explicitly initialize an instance of the **FtpClient** class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the **FtpClient** class

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Initialize Overload List

FtpClient.Login Method

Login to the remote server.

Overload List

Login to the remote server.

public bool Login();

Login to the remote server.

public bool Login(string,string);

Login to the remote server.

public bool Login(string,string,string);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Login Method ()

Login to the remote server.

[Visual Basic]

```
Overloads Public Function Login() As Boolean
```

[C#]

```
public bool Login();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Login** method identifies the user to the remote server. The value of the **UserName** and **Password** properties will be used to authenticate the client session. If the user name or password is invalid, an error will occur. By default, when a connection is established, the user is automatically authenticated. This method is typically used if you wish to log in as another user during the same session.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Login Overload List

FtpClient.Login Method (String, String)

Login to the remote server.

[Visual Basic]

```
Overloads Public Function Login( _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]

```
public bool Login(
   string userName,
   string userPassword
);
```

Parameters

userName
A string that specifies the name of the user logging into the server.

userPassword
A string that specifies the password used to authenticate the user.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Login** method identifies the user to the remote server. If the user name or password is invalid, an error will occur. By default, when a connection is established, the **UserName** and **Password** properties are used to automatically log the user in to the server. This method is typically used if you wish to log in as another user during the same session.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Login Overload List

FtpClient.Login Method (String, String, String)

Login to the remote server.

[Visual Basic]

```
Overloads Public Function Login( _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal userAccount As String _
) As Boolean
```

[C#]

```
public bool Login(
   string userName,
   string userPassword,
   string userAccount
);
```

Parameters

userName
A string that specifies the name of the user logging into the server.

userPassword
A string that specifies the password used to authenticate the user.

userAccount
A string that specifies the account name to be used when authenticating the user.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Login** method identifies the user to the remote server. If the user name or password is invalid, an error will occur. By default, when a connection is established, the **UserName**, **Password** and **Account** properties are used to automatically log the user in to the server. This method is typically used if you wish to log in as another user during the same session.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Login Overload List

FtpClient.Logout Method

Log the current user off the server.

[Visual Basic]

```
Public Function Logout() As Boolean
```

[C#]

```
public bool Logout();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Logout** method logs the current user off the server. The **Login** method may then be used to login as another user during the same session. Note that this method will not terminate the connection with the server.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.MakeDirectory Method

Create a new directory on the server.

[Visual Basic]

```
Public Function MakeDirectory( _
   ByVal pathName As String _
) As Boolean
```

[C#]

```
public bool MakeDirectory(
   string pathName
);
```

Parameters

pathName

A string that specifies the name of the directory to create on the server. The naming and pathing conventions used for the directory must be compatible with what is used on the operating system that hosts the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

Servers may not support creating multiple subdirectories in a single call, so applications should not assume that this can be done. For example, an error may be returned by the server if the new directory name "/Projects/Today" is specified, but the "/Projects" directory does not already exist.

It is also important to note that files and directories on UNIX based systems are case sensitive, so the directory names "Projects" and "projects" refer to two different directories. This is not the case on Windows systems, where either name would refer to the same directory.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.OpenDirectory Method

Open the current working directory on the server.

Overload List

Open the current working directory on the server.

public bool OpenDirectory();

Open the specified directory on the server.

public bool OpenDirectory(string);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.OpenDirectory Method ()

Open the current working directory on the server.

[Visual Basic]

```
Overloads Public Function OpenDirectory() As Boolean
```

[C#]

```
public bool OpenDirectory();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **OpenDirectory** method opens the current working directory on the server so that the list of files in that directory may be obtained using the **GetFirstFile** and **GetNextFile** methods. Once all of the files in the directory have been read, the application must call the **CloseDirectory** method in order to close the data channel to the server. Failure to do this will result in an error the next time the application attempts to transfer a file or open another directory.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.OpenDirectory Overload List

FtpClient.OpenDirectory Method (String)

Open the specified directory on the server.

[Visual Basic]

```
Overloads Public Function OpenDirectory( _
   ByVal pathName As String _
) As Boolean
```

[C#]

```
public bool OpenDirectory(
   string pathName
);
```

Parameters

pathName

A string that specifies the name of the directory to open on the server. The naming and pathing conventions used for the directory must be compatible with what is used on the operating system that hosts the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **OpenDirectory** method opens the specified directory on the server so that the list of files in that directory may be obtained using the **GetFirstFile** and **GetNextFile** methods. Once all of the files in the directory have been read, the application must call the **CloseDirectory** method in order to close the data channel to the server. Failure to do this will result in an error the next time the application attempts to transfer a file or open another directory.

Note that files and directories on UNIX based systems are case sensitive, so the directory names "Projects" and "projects" refer to two different directories. This is not the case on Windows systems, where either name would refer to the same directory.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.OpenDirectory Overload List

FtpClient.OpenFile Method

Open an existing file for reading on the server.

Overload List

Open an existing file for reading on the server.

public bool OpenFile(string);

Open an existing file or create a new file on the server.

public bool OpenFile(string,FtpOpenMode);

Open an existing file or create a new file on the server.

public bool OpenFile(string,FtpOpenMode,long);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.OpenFile Method (String)

Open an existing file for reading on the server.

[Visual Basic]

```
Overloads Public Function OpenFile( _
   ByVal remoteFile As String _
) As Boolean
```

[C#]

```
public bool OpenFile(
   string remoteFile
);
```

Parameters

remoteFile

A string that specifies the name of the file on the server. The filename cannot contain any wildcard characters and must follow the naming conventions of the operating system the server is hosted on.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **OpenFile** method opens an existing file on the server for reading. The **Read** method may then be used to read data from the file. Once the all of the data has been read, the **CloseFile** method must be called to close the data channel.

It is strongly recommended that most applications use the **GetFile** or **PutFile** methods to perform file transfers. These methods are easier to use, and have internal optimizations that improves the overall data transfer rate when compared to implementing the file transfer code in your own application.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.OpenFile Overload List

FtpClient.OpenFile Method (String, FtpOpenMode)

Open an existing file or create a new file on the server.

[Visual Basic]

```
Overloads Public Function OpenFile( _
   ByVal remoteFile As String, _
   ByVal openMode As FtpOpenMode _
) As Boolean
```

[C#]

```
public bool OpenFile(
   string remoteFile,
   FtpOpenMode openMode
);
```

Parameters

remoteFile

A string that specifies the name of the file on the server. The filename cannot contain any wildcard characters and must follow the naming conventions of the operating system the server is hosted on.

openMode

An FtpOpenMode enumeration which specifies how the file should be accessed on the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **OpenFile** method opens an existing file or creates a new file on the remote server using the specified file name. The **Read** method may then be used to read data from the file and the **Write** method may be used to write data to the file. Once the all of the data has been read or written, the **CloseFile** method must be called to close the data channel.

It is strongly recommended that most applications use the **GetFile** or **PutFile** methods to perform file transfers. These methods are easier to use, and have internal optimizations that improves the overall data transfer rate when compared to implementing the file transfer code in your own application.

When a file is created on the remote server, the file ownership and access rights are determined by the server. Some servers may provide a method to change these attributes through site-specific commands. Refer to the server's operating system documentation for more information about what commands may be available.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.OpenFile Overload List

FtpClient.OpenFile Method (String, FtpOpenMode, Int64)

Open an existing file or create a new file on the server.

[Visual Basic]

```
Overloads Public Function OpenFile( _
   ByVal remoteFile As String, _
   ByVal openMode As FtpOpenMode, _
   ByVal offset As Long _
) As Boolean
```

[C#]

```
public bool OpenFile(
   string remoteFile,
   FtpOpenMode openMode,
   long offset
);
```

Parameters

remoteFile
: A string that specifies the name of the file on the server. The filename cannot contain any wildcard characters and must follow the naming conventions of the operating system the server is hosted on.

openMode
: An FtpOpenMode enumeration which specifies how the file should be accessed on the server.

offset
: An integer value which specifies the byte offset where the file transfer should begin. If this argument is omitted, this specifies that the file transfer should start at the beginning of the file. A value greater than zero is typically used to restart a transfer that has not completed successfully. Note that specifying a non-zero offset requires that the server support the REST command to restart transfers.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **OpenFile** method opens an existing file or creates a new file on the remote server using the specified file name. The **Read** method may then be used to read data from the file and the **Write** method may be used to write data to the file. Once the all of the data has been read or written, the **CloseFile** method must be called to close the data channel.

It is strongly recommended that most applications use the **GetFile** or **PutFile** methods to perform file transfers. These methods are easier to use, and have internal optimizations that improves the overall data transfer rate when compared to implementing the file transfer code in your own application.

When a file is created on the remote server, the file ownership and access rights are determined by the server. Some servers may provide a method to change these attributes through site-specific commands. Refer to the server's operating system documentation for more information about what commands may be available.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.OpenFile Overload List

FtpClient.PutData Method

Transfers data from a byte array and stores it in a file on the remote server.

Overload List

Transfers data from a byte array and stores it in a file on the remote server.

public bool PutData(string,byte[],int);

Transfers data from a MemoryStream and stores it in a file on the remote server.

public bool PutData(string,MemoryStream);

Transfers data from a string buffer and stores it in a file on the remote server.

public bool PutData(string,string);

Transfers data from a string buffer and stores it in a file on the remote server.

public bool PutData(string,string,int);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.PutData Method (String, Byte[], Int32)

Transfers data from a byte array and stores it in a file on the remote server.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal remoteFile As String, _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool PutData(
   string remoteFile,
   byte[] buffer,
   int length
);
```

Parameters

remoteFile
A string that specifies the file on the remote system that will contain the data being transferred. If the file already exists, it will be overwritten. The file pathing and name conventions must be that of the remote host.

buffer
A byte array that contains the data to be written to the file.

length
An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the byte array specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutData** method transfers data from a local buffer and stores it on a file on the server. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.PutData Overload List

FtpClient.PutData Method (String, String)

Transfers data from a string buffer and stores it in a file on the remote server.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal remoteFile As String, _
   ByVal buffer As String _
) As Boolean
```

[C#]

```
public bool PutData(
   string remoteFile,
   string buffer
);
```

Parameters

remoteFile

A string that specifies the file on the remote system that will contain the data being transferred. If the file already exists, it will be overwritten. The file pathing and name conventions must be that of the remote host.

buffer

A string that contains the data to be written to the file.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutData** method transfers data from a local buffer and stores it on a file on the server. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

This implementation of the method should only be used with text files.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.PutData Overload List

FtpClient.PutData Method (String, String, Int32)

Transfers data from a string buffer and stores it in a file on the remote server.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal remoteFile As String, _
   ByVal buffer As String, _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool PutData(
   string remoteFile,
   string buffer,
   int length
);
```

Parameters

remoteFile

A string that specifies the file on the remote system that will contain the data being transferred. If the file already exists, it will be overwritten. The file pathing and name conventions must be that of the remote host.

buffer

A string that contains the data to be written to the file.

length

An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the byte array specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutData** method transfers data from a local buffer and stores it on a file on the server. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

This implementation of the method should only be used with text files.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.PutData Overload List

FtpClient.PutData Method (String, MemoryStream)

Transfers data from a MemoryStream and stores it in a file on the remote server.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal remoteFile As String, _
   ByVal memStream As MemoryStream _
) As Boolean
```

[C#]

```
public bool PutData(
   string remoteFile,
   MemoryStream memStream
);
```

Parameters

remoteFile
A string that specifies the file on the remote system that will contain the data being transferred. If the file already exists, it will be overwritten. The file pathing and name conventions must be that of the remote host.

memStream
A MemoryStream that contains the data to be uploaded to the server. This stream must be open and readable.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutData** method transfers data from a MemoryStream and stores it on a file on the server. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The complete contents of the stream will be uploaded to the server, and when the method returns, the current position will be at the end of the stream. The stream must be open and readable, otherwise this method will throw **System.NotSupportedException**.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.PutData Overload List

FtpClient.PutFile Method

Upload a file from the local system to the server.

Overload List

Upload a file from the local system to the server.

public bool PutFile(string);

Upload a file from the local system to the server.

public bool PutFile(string,string);

Upload a file from the local system to the server.

public bool PutFile(string,string,FtpTransferOptions);

Upload a file from the local system to the server.

public bool PutFile(string,string,FtpTransferOptions,long);

See Also

FtpClient Class | SocketTools Namespace | RemoteFile Property | URL Property

FtpClient.PutFile Method (String)

Upload a file from the local system to the server.

[Visual Basic]

```
Overloads Public Function PutFile( _
   ByVal localFile As String _
) As Boolean
```

[C#]

```
public bool PutFile(
   string localFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutFile** method copies an existing file from the local system to the server. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

This implementation of the **PutFile** method is most commonly used when the name of the remote file has already been specified, either by setting the **URL** property or by explicitly setting the **RemoteFile** property.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.PutFile Overload List | RemoteFile Property | URL Property

FtpClient.PutFile Method (String, String)

Upload a file from the local system to the server.

[Visual Basic]

```
Overloads Public Function PutFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String _
) As Boolean
```

[C#]

```
public bool PutFile(
   string localFile,
   string remoteFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

remoteFile

A string that specifies the file on the server that will be created. If the file already exists on the server, it will be overwritten. The file pathing and name conventions must be that of the remote host.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutFile** method copies an existing file from the local system to the server. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.PutFile Overload List

FtpClient.PutFile Method (String, String, FtpTransferOptions)

Upload a file from the local system to the server.

[Visual Basic]

```
Overloads Public Function PutFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal options As FtpTransferOptions _
) As Boolean
```

[C#]

```
public bool PutFile(
   string localFile,
   string remoteFile,
   FtpTransferOptions options
);
```

Parameters

localFile

A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

remoteFile

A string that specifies the file on the server that will be created, overwritten or appended to. The file pathing and name conventions must be that of the remote host.

options

An FtpTransferOptions enumeration which specifies how the file should be uploaded to the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutFile** method copies an existing file from the local system to the server. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.PutFile Overload List

FtpClient.PutFile Method (String, String, FtpTransferOptions, Int64)

Upload a file from the local system to the server.

[Visual Basic]

```
Overloads Public Function PutFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal options As FtpTransferOptions, _
   ByVal offset As Long _
) As Boolean
```

[C#]

```
public bool PutFile(
   string localFile,
   string remoteFile,
   FtpTransferOptions options,
   long offset
);
```

Parameters

localFile
: A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

remoteFile
: A string that specifies the file on the server that will be created, overwritten or appended to. The file pathing and name conventions must be that of the remote host.

options
: An FtpTransferOptions enumeration which specifies how the file should be uploaded to the server.

offset
: A numeric value which specifies the byte offset where the file transfer should begin. The default value of zero specifies that the file transfer should start at the beginning of the file. A value greater than zero is typically used to restart a transfer that has not completed successfully. Note that specifying a non-zero offset requires that the server support the REST command to restart transfers.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutFile** method copies an existing file from the local system to the server. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.PutFile Overload List

FtpClient.PutMultipleFiles Method

Upload multiple files from the local system to the server using a wildcard mask.

Overload List

Upload multiple files from the local system to the server using a wildcard mask.

public bool PutMultipleFiles(string,string);

Upload multiple files from the local system to the server using a wildcard mask.

public bool PutMultipleFiles(string,string,string);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.PutMultipleFiles Method (String, String, String)

Upload multiple files from the local system to the server using a wildcard mask.

[Visual Basic]

```
Overloads Public Function PutMultipleFiles( _
   ByVal localPath As String, _
   ByVal remotePath As String, _
   ByVal fileMask As String _
) As Boolean
```

[C#]

```
public bool PutMultipleFiles(
   string localPath,
   string remotePath,
   string fileMask
);
```

Parameters

localPath
A string argument which specifies the name of the directory on the local system where the files will be copied from. You must have permission to read the contents of the directory.

remotePath
A string argument which specifies the name of the directory on the remote system where the files will be stored. You must have permission to modify the contents of the directory and create files.

fileMask
A string argument which specifies the wildcard mask to be used when selecting what files should be transferred. An empty string indicates that all files in the specified directory should be uploaded. Typically, this argument is a wildcard mask that limits the files uploaded to the server to those which match a specific extension.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutMultipleFiles** method copies multiple files from the local system to the remote server. If the remote file already exists, it is overwritten. This method will cause the current thread to block until all of the files have been transferred, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.PutMultipleFiles Overload List

FtpClient.PutMultipleFiles Method (String, String)

Upload multiple files from the local system to the server using a wildcard mask.

[Visual Basic]

```
Overloads Public Function PutMultipleFiles( _
   ByVal localPath As String, _
   ByVal remotePath As String _
) As Boolean
```

[C#]

```
public bool PutMultipleFiles(
   string localPath,
   string remotePath
);
```

Parameters

localPath
A string argument which specifies the name of the directory on the local system where the files will be copied from. You must have permission to read the contents of the directory.

remotePath
A string argument which specifies the name of the directory on the remote system where the files will be stored. You must have permission to modify the contents of the directory and create files.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutMultipleFiles** method copies multiple files from the local system to the remote server. If the remote file already exists, it is overwritten. This method will cause the current thread to block until all of the files have been transferred, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.PutMultipleFiles Overload List

FtpClient.Read Method

Read file data from the server and store it in a byte array.

Overload List

Read file data from the server and store it in a byte array.

public int Read(byte[]);

Read file data from the server and store it in a byte array.

public int Read(byte[],int);

Read file data from the server and store it in a string.

public int Read(ref string);

Read file data from the server and store it in a string.

public int Read(ref string,int);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Read Method (Byte[])

Read file data from the server and store it in a byte array.

[Visual Basic]

```
Overloads Public Function Read( _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Read(
   byte[] buffer
);
```

Parameters

buffer

A byte array that the data will be stored in.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the number of bytes specified. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Read Overload List

FtpClient.Read Method (Byte[], Int32)

Read file data from the server and store it in a byte array.

[Visual Basic]

```
Overloads Public Function Read( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Read(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array that the data will be stored in.

length

An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the number of bytes specified. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Read Overload List

FtpClient.Read Method (String)

Read file data from the server and store it in a string.

[Visual Basic]

```
Overloads Public Function Read( _
   ByRef buffer As String _
) As Integer
```

[C#]

```
public int Read(
   ref string buffer
);
```

Parameters

buffer

A string that is passed by reference. When the method returns, it will contain the data read from the server.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the number of bytes specified. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Read Overload List

FtpClient.Read Method (String, Int32)

Read file data from the server and store it in a string.

[Visual Basic]

```
Overloads Public Function Read( _
   ByRef buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Read(
   ref string buffer,
   int length
);
```

Parameters

buffer

A string that is passed by reference. When the method returns, it will contain the data read from the server.

length

An integer value which specifies the maximum number of bytes of data to read.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the number of bytes specified. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Read Overload List

FtpClient.RemoveDirectory Method

Remove a directory on the server.

[Visual Basic]

```
Public Function RemoveDirectory( _
   ByVal pathName As String _
) As Boolean
```

[C#]

```
public bool RemoveDirectory(
   string pathName
);
```

Parameters

pathName

A string that specifies the name of the directory to remove from the server. The naming and pathing conventions used for the directory must be compatible with what is used on the operating system that hosts the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **RemoveDirectory** method removes an existing directory on the remote host. You must have the appropriate permission to remove the directory, or an error will occur. Note that most operating systems will not permit you to remove a directory that contains files or other subdirectories.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.RenameFile Method

Change the name of a file on the server.

[Visual Basic]

```
Public Function RenameFile( _
   ByVal oldName As String, _
   ByVal newName As String _
) As Boolean
```

[C#]

```
public bool RenameFile(
   string oldName,
   string newName
);
```

Parameters

oldName
: A string that specifies the name of the file to be renamed on the server. The file must exist on the server, otherwise an error will be returned.

newName
: A string that specifies the new name for the file on the server. The naming conventions used for the file must be compatible with what is used on the operating system that hosts the server. Note that some servers may not permit you to rename the file if a file with the new name already exists.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **RenameFile** method changes the name of an existing file on the server to a new name. Note that you must have permission to rename the file or an error will occur. On UNIX based systems this means that you must have write permission to the directory where the file is being renamed.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. If a connection to a server has been established, it will be terminated the resources allocated for the client session will be released. All properties will be reset to their default values.

The **Reset** and **Uninitialize** methods will abort all active background transfers and wait for those tasks to complete before returning to the caller. It is recommended that your application explicitly wait for background transfers to complete or abort them using this method before allowing the program to terminate. This will ensure that your program can perform any necessary cleanup operations. If there are active background tasks running at the time that the class instance is disposed, it can force the instance to stop those worker threads immediately without waiting for them to terminate gracefully.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.SetFilePermissions Method

Change the access permissions for a file on the server.

[Visual Basic]

```
Public Function SetFilePermissions( _
   ByVal remoteFile As String, _
   ByVal filePerms As FtpPermissions _
) As Boolean
```

[C#]

```
public bool SetFilePermissions(
   string remoteFile,
   FtpPermissions filePerms
);
```

Parameters

remoteFile

A string that specifies the name of the file that the access permissions are to be returned for. The filename cannot contain any wildcard characters.

filePerms

An FtpPermissions enumeration which specifies the new permissions for the file.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method uses the SITE CHMOD command to set the permissions for the file. This command is typically only supported on servers that are hosted on UNIX based systems. If the command is not supported, an error will be returned. You can use the **Features** property to determine what features are available and/or enabled on the server.

Users who are familiar with the UNIX operating system will recognize the **chmod** command used to change the file permissions. However, it should be noted that the numeric value used as an argument to the command is in octal, not decimal. For example, issuing the command **chmod 644 filename.txt** on a UNIX based system will make the file readable and writable by the owner, and readable by other users in the owner's group as well as all other users. The value 644 is an octal value, which is equivalent to the decimal value 420. If you were to mistakenly specify 644 as the value for the *Permissions* argument, rather than the decimal value of 420, the permissions on the file would be incorrect. It is strongly recommended that you use the enumeration values and do not cast a numeric value as the argument.

Note that Visual Basic allows you to specify an integer value in octal by prefixing it with &O. For example, &O644 could be used as the file permissions value. C# and C++ consider any integer with a preceding 0 to be an octal number, so 0644 would be a valid permissions value.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.SetFileTime Method

Changes the modification date and time for a file on the server.

[Visual Basic]

```
Public Function SetFileTime( _
   ByVal remoteFile As String, _
   ByVal fileDate As Date _
) As Boolean
```

[C#]

```
public bool SetFileTime(
   string remoteFile,
   DateTime fileDate
);
```

Parameters

remoteFile

A string that specifies the name of the file on the server. The filename cannot contain any wildcard characters and must follow the naming conventions of the operating system the server is hosted on.

fileDate

A **System.DateTime** value that specifies the new date and time for the file.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SetFileTime** method changes the modification date and time for the specified file on the remote server. This method uses the MTDM command to change the modification time for the file. If the server does not support this command, the method will return an error. Note that some servers only support the MDTM command to return, but not change, the file modification time.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.TaskAbort Method

Abort all asynchronous tasks that are currently active.

Overload List

Abort all asynchronous tasks that are currently active.

public bool TaskAbort();

Abort the specified asynchronous task.

public bool TaskAbort(int);

Abort the specified asynchronous task.

public bool TaskAbort(int,int);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.TaskAbort Method ()

Abort all asynchronous tasks that are currently active.

[Visual Basic]

```
Overloads Public Function TaskAbort() As Boolean
```

[C#]

```
public bool TaskAbort();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskAbort** method signals all background worker threads created by this instance of the class to abort their current operation and terminate as soon as possible. This version of the method will signal each active task and return immediately to the caller.

The **Reset** and **Uninitialize** methods will abort all active background transfers and wait for those tasks to complete before returning to the caller. It is recommended that your application explicitly wait for background transfers to complete or abort them using this method before allowing the program to terminate. This will ensure that your program can perform any necessary cleanup operations. If there are active background tasks running at the time that the class instance is disposed, it can force the instance to stop those worker threads immediately without waiting for them to terminate gracefully.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.TaskAbort Overload List

FtpClient.TaskAbort Method (Int32)

Abort the specified asynchronous task.

[Visual Basic]

```
Overloads Public Function TaskAbort( _
   ByVal taskId As Integer _
) As Boolean
```

[C#]

```
public bool TaskAbort(
   int taskId
);
```

Parameters

taskId

An integer value that specifies the unique identifier associated with a background task.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskAbort** method signals the background worker thread associated with the task ID to abort the current operation and terminate as soon as possible. This version of the method returns immediately after the background thread has been signaled.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.TaskAbort Overload List

FtpClient.TaskAbort Method (Int32, Int32)

Abort the specified asynchronous task.

[Visual Basic]

```
Overloads Public Function TaskAbort( _
   ByVal taskId As Integer, _
   ByVal timeWait As Integer _
) As Boolean
```

[C#]

```
public bool TaskAbort(
   int taskId,
   int timeWait
);
```

Parameters

taskId
An integer value that specifies the unique identifier associated with a background task.

timeWait
An integer value that specifies the number of milliseconds to wait for the background task to abort.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskAbort** method signals the background worker thread associated with the task ID to abort the current operation and terminate as soon as possible. If the ***timeWait*** parameter has a value of zero, the method returns immediately after the background thread has been signaled. If the ***timeWait*** parameter is non-zero, the method will wait that amount of time for the background thread to terminate.

The **Reset** and **Uninitialize** methods will abort all active background transfers and wait for those tasks to complete before returning to the caller. It is recommended that your application explicitly wait for background transfers to complete or abort them using this method before allowing the program to terminate. This will ensure that your program can perform any necessary cleanup operations. If there are active background tasks running at the time that the class instance is disposed, it can force the instance to stop those worker threads immediately without waiting for them to terminate gracefully.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.TaskAbort Overload List

FtpClient.TaskDone Method

Determine if the current asynchronous task has completed.

Overload List

Determine if the current asynchronous task has completed.

public bool TaskDone();

Determine if an asynchronous task has completed.

public bool TaskDone(int);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.TaskDone Method ()

Determine if the current asynchronous task has completed.

[Visual Basic]

```
Overloads Public Function TaskDone() As Boolean
```

[C#]

```
public bool TaskDone();
```

Return Value

This method returns a Boolean value. If the task has finished, the return value is **true**. If the background task is still active, the return value is **false**.

Remarks

The **TaskDone** method is used to determine if the current asynchronous task has completed. This overloaded version of the method is functionally equivalent to providing the value of the **TaskId** property as the unique task identifier.

If you use this method to poll the status of a background task from within the main UI thread, you must ensure that Windows messages are processed so that the application remains responsive to the end-user. To check if a background transfer has completed, it is recommended that you use a timer to periodically call this method rather than calling it repeatedly within a loop.

To determine if the task completed successfully, the **TaskWait** method will provide the last error code associated with the task. Note that if this method returns **true**, it is guaranteed that calling **TaskWait** using the same task ID will return the error code to the caller immediately without causing the current thread to block.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.TaskDone Overload List

FtpClient.TaskDone Method (Int32)

Determine if an asynchronous task has completed.

[Visual Basic]

```
Overloads Public Function TaskDone( _
   ByVal taskId As Integer _
) As Boolean
```

[C#]

```
public bool TaskDone(
   int taskId
);
```

Parameters

taskId

An optional integer value that specifies the unique identifier associated with a background task.

Return Value

This method returns a Boolean value. If the task has finished, the return value is **true**. If the background task is still active, the return value is **false**.

Remarks

The **TaskDone** method is used to determine if the specified asynchronous task has completed.

If you use this method to poll the status of a background task from within the main UI thread, you must ensure that Windows messages are processed so that the application remains responsive to the end-user. To check if a background transfer has completed, it is recommended that you use a timer to periodically call this method rather than calling it repeatedly within a loop.

To determine if the task completed successfully, the **TaskWait** method will provide the last error code associated with the task. Note that if this method returns **true**, it is guaranteed that calling **TaskWait** using the same task ID will return the error code to the caller immediately without causing the current thread to block.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.TaskDone Overload List

FtpClient.TaskResume Method

Resume execution of the current asynchronous task.

Overload List

Resume execution of the current asynchronous task.

public bool TaskResume();

Resume execution of an asynchronous task.

public bool TaskResume(int);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.TaskResume Method ()

Resume execution of the current asynchronous task.

[Visual Basic]

```
Overloads Public Function TaskResume() As Boolean
```

[C#]

```
public bool TaskResume();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskResume** method resumes execution of the current background task that was previously suspended using the **TaskSuspend** method. This overloaded version of the method is functionally equivalent to providing the value of the **TaskId** property as the unique task identifier.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.TaskResume Overload List

FtpClient.TaskResume Method (Int32)

Resume execution of an asynchronous task.

[Visual Basic]

```
Overloads Public Function TaskResume( _
   ByVal taskId As Integer _
) As Boolean
```

[C#]

```
public bool TaskResume(
   int taskId
);
```

Parameters

taskId

An integer value that specifies the unique identifier associated with a background task.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskResume** method resumes execution of the background worker thread that was previously suspended using the **TaskSuspend** method.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.TaskResume Overload List

FtpClient.TaskSuspend Method

Suspend execution of the current asynchronous task.

Overload List

Suspend execution of the current asynchronous task.

public bool TaskSuspend();

Suspend execution of an asynchronous task.

public bool TaskSuspend(int);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.TaskSuspend Method ()

Suspend execution of the current asynchronous task.

[Visual Basic]

```
Overloads Public Function TaskSuspend() As Boolean
```

[C#]

```
public bool TaskSuspend();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskSuspend** method will suspend execution of the background worker thread associated with the current task. This overloaded version of the method is functionally equivalent to providing the value of the **TaskId** property as the unique task identifier.

Once the task has been suspended, it will no longer be scheduled for execution, however the client session will remain active and the task may be resumed using the **TaskResume** method. Note that if a task is suspended for a long period of time, the background operation may fail because it has exceeded the timeout period imposed by the server.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.TaskSuspend Overload List

FtpClient.TaskSuspend Method (Int32)

Suspend execution of an asynchronous task.

[Visual Basic]

```
Overloads Public Function TaskSuspend( _
   ByVal taskId As Integer _
) As Boolean
```

[C#]

```
public bool TaskSuspend(
   int taskId
);
```

Parameters

taskId

An integer value that specifies the unique identifier associated with a background task.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskSuspend** method will suspend execution of the background worker thread associated with the task.

Once the task has been suspended, it will no longer be scheduled for execution, however the client session will remain active and the task may be resumed using the **TaskResume** method. Note that if a task is suspended for a long period of time, the background operation may fail because it has exceeded the timeout period imposed by the server.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.TaskSuspend Overload List

FtpClient.TaskWait Method

Wait for all asynchronous tasks to complete.

Overload List

Wait for all asynchronous tasks to complete.

public bool TaskWait();

Wait for an asynchronous task to complete.

public bool TaskWait(int);

Wait for an asynchronous task to complete.

public bool TaskWait(int,int);

Wait for an asynchronous task to complete.

public bool TaskWait(int,int,ref ErrorCode);

Wait for an asynchronous task to complete.

public bool TaskWait(int,int,ref int,ref ErrorCode);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.TaskWait Method ()

Wait for all asynchronous tasks to complete.

[Visual Basic]

```
Overloads Public Function TaskWait() As Boolean
```

[C#]

```
public bool TaskWait();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This overloaded version of the **TaskWait** method will cause the current working thread to block until all background tasks created by this instance of the class have completed. If there are no active background tasks, this method will return to the caller immediately.

You should not call this version of the method from the main UI thread. Windows messages will not be processed while this method is blocked waiting for the background tasks to complete, and this can cause your application to appear non-responsive to the end-user. If you have a GUI application and you need to determine if all tasks have completed, create a timer to periodically check the value of the **TaskCount** property. When it returns zero, there are no active background tasks executing.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.TaskWait Overload List

FtpClient.TaskWait Method (Int32)

Wait for an asynchronous task to complete.

[Visual Basic]

```
Overloads Public Function TaskWait( _
   ByVal taskId As Integer _
) As Boolean
```

[C#]

```
public bool TaskWait(
   int taskId
);
```

Parameters

taskId

An integer value that specifies the unique identifier associated with a background task.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskWait** method waits for the specified task to complete. This method will cause the current working thread to block for an indefinite period of time until the task completes. If the specified task has already completed at the time this method is called, the method will return immediately without causing the current thread to block.

You should not call this overloaded version of the method from the main UI thread. Windows messages will not be processed while this method is blocked waiting for the background task to complete, and this can cause your application to appear non-responsive to the end-user. If you have a GUI application and you need to determine if a background task has finished, create a timer to periodically call the **TaskDone** method. When it returns **true** (indicating that the task has completed), you can safely call **TaskWait** to obtain the elapsed time and last error code without blocking the current thread.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.TaskWait Overload List

FtpClient.TaskWait Method (Int32, Int32)

Wait for an asynchronous task to complete.

[Visual Basic]

```
Overloads Public Function TaskWait( _
   ByVal taskId As Integer, _
   ByVal timeWait As Integer _
) As Boolean
```

[C#]

```
public bool TaskWait(
   int taskId,
   int timeWait
);
```

Parameters

taskId

An integer value that specifies the unique identifier associated with a background task.

timeWait

An integer value that specifies the number of milliseconds to wait for the background task to complete.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskWait** method waits for the specified task to complete. This method will cause the current working thread to block until the task completes or the amount of time exceeds the number of milliseconds specified by the caller. If the *timeWait* parameter is zero, then this method will poll the status of the task and return immediately to the caller. If the specified task has already completed at the time this method is called, the method will return immediately without causing the current thread to block.

You should not call this method from the main UI thread with a long timeout period to wait for a background task to complete. Windows messages will not be processed while this method is blocked waiting for the background task to complete, and this can cause your application to appear non-responsive to the end-user. If you have a GUI application and you need to determine if a background task has finished, create a timer to periodically call the **TaskDone** method. When it returns **true** (indicating that the task has completed), you can safely call **TaskWait** to obtain the elapsed time and last error code without blocking the current thread.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.TaskWait Overload List

FtpClient.TaskWait Method (Int32, Int32, ErrorCode)

Wait for an asynchronous task to complete.

[Visual Basic]

```
Overloads Public Function TaskWait( _
   ByVal taskId As Integer, _
   ByVal timeWait As Integer, _
   ByRef taskError As ErrorCode _
) As Boolean
```

[C#]

```
public bool TaskWait(
   int taskId,
   int timeWait,
   ref ErrorCode taskError
);
```

Parameters

taskId

An integer value that specifies the unique identifier associated with a background task.

timeWait

An integer value that specifies the number of milliseconds to wait for the background task to complete.

taskError

An ErrorCode value passed by reference that will contain the last error code set by the asynchronous task.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskWait** method waits for the specified task to complete. This method will cause the current working thread to block until the task completes or the amount of time exceeds the number of milliseconds specified by the caller. If the *timeWait* parameter is zero, then this method will poll the status of the task and return immediately to the caller.

If the specified task has already completed at the time this method is called, the method will return immediately without causing the current thread to block. The ***taskError*** parameter will contain the last error code value that was set by the worker thread before it terminated. If the ***taskError*** value is zero, that means that the background task was successful and no error occurred. A non-zero value will indicate that the background task has failed.

You should not call this method from the main UI thread with a long timeout period to wait for a background task to complete. Windows messages will not be processed while this method is blocked waiting for the background task to complete, and this can cause your application to appear non-responsive to the end-user. If you have a GUI application and you need to determine if a background task has finished, create a timer to periodically call the **TaskDone** method. When it returns **true** (indicating that the task has completed), you can safely call **TaskWait** to obtain the elapsed time and last error code without blocking the current thread.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.TaskWait Overload List

FtpClient.TaskWait Method (Int32, Int32, Int32, ErrorCode)

Wait for an asynchronous task to complete.

[Visual Basic]

```
Overloads Public Function TaskWait( _
   ByVal taskId As Integer, _
   ByVal timeWait As Integer, _
   ByRef timeElapsed As Integer, _
   ByRef taskError As ErrorCode _
) As Boolean
```

[C#]

```
public bool TaskWait(
   int taskId,
   int timeWait,
   ref int timeElapsed,
   ref ErrorCode taskError
);
```

Parameters

taskId
An integer value that specifies the unique identifier associated with a background task.

timeWait
An integer value that specifies the number of milliseconds to wait for the background task to complete.

timeElapsed
An integer value passed by reference that will contain the elapsed time for the task in milliseconds when the method returns.

taskError
An ErrorCode value passed by reference that will contain the last error code set by the asynchronous task.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskWait** method waits for the specified task to complete. This method will cause the current working thread to block until the task completes or the amount of time exceeds the number of milliseconds specified by the caller. If the *timeWait* parameter is zero, then this method will poll the status of the task and return immediately to the caller.

If the specified task has already completed at the time this method is called, the method will return immediately without causing the current thread to block. The *timeElapsed* parameter contain the number of milliseconds that it took for the task to complete. The *taskError* parameter will contain the last error code value that was set by the worker thread before it terminated. If the *taskError* value is zero, that means that the background task was successful and no error occurred. A non-zero value will indicate that the background task has failed.

You should not call this method from the main UI thread with a long timeout period to wait for a background task to complete. Windows messages will not be processed while this method is blocked

waiting for the background task to complete, and this can cause your application to appear non-responsive to the end-user. If you have a GUI application and you need to determine if a background task has finished, create a timer to periodically call the **TaskDone** method. When it returns **true** (indicating that the task has completed), you can safely call **TaskWait** to obtain the elapsed time and last error code without blocking the current thread.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.TaskWait Overload List

FtpClient.Uninitialize Method

Uninitialize the class library and release any resources allocated for the current thread.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method terminates any active connection, releases resources allocated for the current thread and unloads the networking library. After this method has been called, no further network operations may be performed until the class instance has been re-initialized.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the class is no longer needed.

The **Reset** and **Uninitialize** methods will abort all active background transfers and wait for those tasks to complete before returning to the caller. It is recommended that your application explicitly wait for background transfers to complete or abort them using this method before allowing the program to terminate. This will ensure that your program can perform any necessary cleanup operations. If there are active background tasks running at the time that the class instance is disposed, it can force the instance to stop those worker threads immediately without waiting for them to terminate gracefully.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.VerifyFile Method

Verify that the contents of a file on the local system are the same as the specified file on the server.

Overload List

Verify that the contents of a file on the local system are the same as the specified file on the server.

public bool VerifyFile(string,string);

Verify that the contents of a file on the local system are the same as the specified file on the server.

public bool VerifyFile(string,string,FtpVerifyOptions);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.VerifyFile Method (String, String)

Verify that the contents of a file on the local system are the same as the specified file on the server.

[Visual Basic]

```
Overloads Public Function VerifyFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String _
) As Boolean
```

[C#]

```
public bool VerifyFile(
   string localFile,
   string remoteFile
);
```

Parameters

localFile
A string value which specifies the name of the local file.

remoteFile
A string value which specifies the name of the file on the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **VerifyFile** method will attempt to verify that the contents of the local and remote files are identical using one of several methods, based on the features that the server supports. Preference will be given to the most reliable method available, using either an MD5 hash, a CRC32 checksum or comparing the size of the file, in that order.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.VerifyFile Overload List

FtpClient.VerifyFile Method (String, String, FtpVerifyOptions)

Verify that the contents of a file on the local system are the same as the specified file on the server.

[Visual Basic]

```
Overloads Public Function VerifyFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal options As FtpVerifyOptions _
) As Boolean
```

[C#]

```
public bool VerifyFile(
   string localFile,
   string remoteFile,
   FtpVerifyOptions options
);
```

Parameters

localFile
A string value which specifies the name of the local file.

remoteFile
A string value which specifies the name of the file on the server.

options
An FtpVerifyOptions enumeration which specifies one or more options that should be used when comparing the files.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **VerifyFile** method will attempt to verify that the contents of the local and remote files are identical using one of several methods, based on the features that the server supports. Preference will be given to the most reliable method available, using either an MD5 hash, a CRC32 checksum or comparing the size of the file, in that order.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.VerifyFile Overload List

FtpClient.Write Method

Write one or more bytes of data to the server.

Overload List

Write one or more bytes of data to the server.

public int Write(byte[]);

Write one or more bytes of data to the server.

public int Write(byte[],int);

Write a string of characters to the server.

public int Write(string);

Write a string of characters to the server.

public int Write(string,int);

See Also

FtpClient Class | SocketTools Namespace

FtpClient.Write Method (Byte[])

Write one or more bytes of data to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Write(
   byte[] buffer
);
```

Parameters

buffer
    A byte array that contains the data to be written to the server.

Return Value

An integer value which specifies the number of bytes actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to the server. If there is enough room in the socket's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the socket is in blocking mode, then the method will block until the data can be sent. If the socket is in non-blocking mode and the send buffer is full, an error will occur.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Write Overload List

FtpClient.Write Method (Byte[], Int32)

Write one or more bytes of data to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Write(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array that contains the data to be written to the server.

length

An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to the server. If there is enough room in the socket's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the socket is in blocking mode, then the method will block until the data can be sent. If the socket is in non-blocking mode and the send buffer is full, an error will occur.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Write Overload List

FtpClient.Write Method (String)

Write a string of characters to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String _
) As Integer
```

[C#]

```
public int Write(
   string buffer
);
```

Parameters

buffer
A string which contains the data to be written to the server.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to the server. If there is enough room in the socket's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the socket is in blocking mode, then the method will block until the data can be sent. If the socket is in non-blocking mode and the send buffer is full, an error will occur.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Write Overload List

FtpClient.Write Method (String, Int32)

Write a string of characters to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Write(
   string buffer,
   int length
);
```

Parameters

buffer

A string which contains the data to be written to the server.

length

An integer value which specifies the maximum number of characters to write. This value cannot be larger than the length of the string specified by the caller.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to the server. If there is enough room in the socket's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the socket is in blocking mode, then the method will block until the data can be sent. If the socket is in non-blocking mode and the send buffer is full, an error will occur.

See Also

FtpClient Class | SocketTools Namespace | FtpClient.Write Overload List

FtpClient Events

The events of the **FtpClient** class are listed below. For a complete list of **FtpClient** class members, see the FtpClient Members topic.

Public Instance Events

⚡ OnCancel	Occurs when a blocking client operation is canceled.
⚡ OnCommand	Occurs when the client sends a command to the remote host and receives a reply indicating the result of that command.
⚡ OnConnect	Occurs when a connection is established with the remote host.
⚡ OnDisconnect	Occurs when the remote host disconnects from the local system.
⚡ OnError	Occurs when an socket operation fails.
⚡ OnFileList	Occurs when a directory listing is parsed by the class.
⚡ OnGetFile	Occurs when a file download has been initiated.
⚡ OnLastFile	Occurs when the last file in a directory listing has been processed.
⚡ OnProgress	Occurs as a data stream is being read or written to the socket.
⚡ OnPutFile	Occurs when a file upload is initiated.
⚡ OnRead	Occurs when data is available to be read from the socket.
⚡ OnTaskBegin	Occurs when an asynchronous task begins execution.
⚡ OnTaskEnd	Occurs when an asynchronous task completes.
⚡ OnTaskRun	Occurs while a background task is active.
⚡ OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
⚡ OnWrite	Occurs when data can be written to the socket.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.OnCancel Event

Occurs when a blocking client operation is canceled.

[Visual Basic]
```
Public Event OnCancel As EventHandler
```

[C#]
```
public event EventHandler OnCancel;
```

Remarks

The **OnCancel** event is generated when a blocking client operation, such as sending or receiving data, is canceled with the **Cancel** method. To assist in determining which operation was canceled, check the value of the **Status** property.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.OnCommand Event

Occurs when the client sends a command to the remote host and receives a reply indicating the result of that command.

[Visual Basic]

```
Public Event OnCommand As OnCommandEventHandler
```

[C#]

```
public event OnCommandEventHandler OnCommand;
```

Event Data

The event handler receives an argument of type FtpClient.CommandEventArgs containing data related to this event. The following **FtpClient.CommandEventArgs** properties provide information specific to this event.

Property	Description
ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.

Remarks

The **OnCommand** event is generated when the client receives a reply from the server after some action has been taken.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.CommandEventArgs Class

Provides data for the OnCommand event.

For a list of all members of this type, see FtpClient.CommandEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FtpClient.CommandEventArgs

[Visual Basic]

```
Public Class FtpClient.CommandEventArgs
    Inherits EventArgs
```

[C#]

```
public class FtpClient.CommandEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

CommandEventArgs specifies the result code and result string for the last command executed by the server.

The OnCommand event occurs whenever a command is executed on the server.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

FtpClient.CommandEventArgs Members | SocketTools Namespace

FtpClient.CommandEventArgs Members

FtpClient.CommandEventArgs overview

Public Instance Constructors

FtpClient.CommandEventArgs Constructor	Initializes a new instance of the FtpClient.CommandEventArgs class.

Public Instance Properties

ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpClient.CommandEventArgs Class | SocketTools Namespace

FtpClient.CommandEventArgs Constructor

Initializes a new instance of the FtpClient.CommandEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FtpClient.CommandEventArgs();
```

See Also

FtpClient.CommandEventArgs Class | SocketTools Namespace

FtpClient.CommandEventArgs Properties

The properties of the **FtpClient.CommandEventArgs** class are listed below. For a complete list of **FtpClient.CommandEventArgs** class members, see the FtpClient.CommandEventArgs Members topic.

Public Instance Properties

ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.

See Also

FtpClient.CommandEventArgs Class | SocketTools Namespace

FtpClient.CommandEventArgs.ResultCode Property

Gets a value which specifies the last result code returned by the server.

[Visual Basic]

```
Public ReadOnly Property ResultCode As Integer
```

[C#]

```
public int ResultCode {get;}
```

Property Value

An integer value which specifies the last result code returned by the server.

Remarks

This property should be checked after the **Command** method is used to execute a command on the server to determine if the operation was successful. Result codes are three-digit numeric values returned by the remote server and may be broken down into the following ranges:

ResultCode	Description
100-199	Positive preliminary result. This indicates that the requested action is being initiated, and the client should expect another reply from the server before proceeding.
200-299	Positive completion result. This indicates that the server has successfully completed the requested action.
300-399	Positive intermediate result. This indicates that the requested action cannot complete until additional information is provided to the server.
400-499	Transient negative completion result. This indicates that the requested action did not take place, but the error condition is temporary and may be attempted again.
500-599	Permanent negative completion result. This indicates that the requested action did not take place.

It is important to note that while some result codes have become standardized, not all servers respond to commands using the same result codes. For example, one server may respond with a result code of 221 to indicate success, while another may respond with a value of 235. It is recommended that applications check for ranges of values to determine if a command was successful, not a specific value.

See Also

FtpClient.CommandEventArgs Class | SocketTools Namespace

FtpClient.CommandEventArgs.ResultString Property

Gets a string value which describes the result of the previous command.

[Visual Basic]

```
Public ReadOnly Property ResultString As String
```

[C#]

```
public string ResultString {get;}
```

Property Value

A string which describes the result of the previous command executed on the server.

Remarks

This string is generated by the remote server, and typically is used to describe the result code. For example, if an error is indicated by the result code, the result string may describe the condition that caused the error.

See Also

FtpClient.CommandEventArgs Class | SocketTools Namespace

FtpClient.OnConnect Event

Occurs when a connection is established with the remote host.

[Visual Basic]
```
Public Event OnConnect As EventHandler
```

[C#]
```
public event EventHandler OnConnect;
```

Remarks

The **OnConnect** event occurs when a connection is made with a remote host as a result of a **Connect** method call. When the **Connect** method is called and the **Blocking** property is set to **false**, a socket is created but the connection is not actually established until after this event occurs. Between the time connection process is started and this event fires, no operation may be performed by the client other than calling the **Disconnect** method.

This event is only generated if the client is in non-blocking mode.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.OnDisconnect Event

Occurs when the remote host disconnects from the local system.

[Visual Basic]

```
Public Event OnDisconnect As EventHandler
```

[C#]

```
public event EventHandler OnDisconnect;
```

Remarks

The **OnDisconnect** event occurs when the remote host closes its socket, terminating its connection with the application. Because there may still be data in the socket receive buffers, you should continue to read data from the socket until the **Read** method returns a value of 0. Once all of the data has been read, you should call the **Disconnect** method to close the local socket and terminate the session.

This event is only generated if the client is in non-blocking mode.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.OnError Event

Occurs when an socket operation fails.

[Visual Basic]

```
Public Event OnError As OnErrorEventHandler
```

[C#]

```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type FtpClient.ErrorEventArgs containing data related to this event. The following **FtpClient.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a client operation fails.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see FtpClient.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FtpClient.ErrorEventArgs

[Visual Basic]

```
Public Class FtpClient.ErrorEventArgs
    Inherits EventArgs
```

[C#]

```
public class FtpClient.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

FtpClient.ErrorEventArgs Members | SocketTools Namespace

FtpClient.ErrorEventArgs Members

FtpClient.ErrorEventArgs overview

Public Instance Constructors

FtpClient.ErrorEventArgs Constructor	Initializes a new instance of the FtpClient.ErrorEventArgs class.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpClient.ErrorEventArgs Class | SocketTools Namespace

FtpClient.ErrorEventArgs Constructor

Initializes a new instance of the FtpClient.ErrorEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FtpClient.ErrorEventArgs();
```

See Also

FtpClient.ErrorEventArgs Class | SocketTools Namespace

FtpClient.ErrorEventArgs Properties

The properties of the **FtpClient.ErrorEventArgs** class are listed below. For a complete list of **FtpClient.ErrorEventArgs** class members, see the FtpClient.ErrorEventArgs Members topic.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

FtpClient.ErrorEventArgs Class | SocketTools Namespace

FtpClient.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

FtpClient.ErrorEventArgs Class | SocketTools Namespace | Error Property

FtpClient.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]
```
Public ReadOnly Property Error As ErrorCode
```

[C#]
```
public FtpClient.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

FtpClient.ErrorEventArgs Class | SocketTools Namespace | Description Property

FtpClient.OnGetFile Event

Occurs when a file download has been initiated.

[Visual Basic]

```
Public Event OnGetFile As OnGetFileEventHandler
```

[C#]

```
public event OnGetFileEventHandler OnGetFile;
```

Event Data

The event handler receives an argument of type FtpClient.GetFileEventArgs containing data related to this event. The following **FtpClient.GetFileEventArgs** properties provide information specific to this event.

Property	Description
LocalFile	Gets a value which specifies the name of the local file.
RemoteFile	Gets a value which specifies the name of the remote file.

Remarks

The **OnGetFile** event is generated when a file transfer is initiated by calling the **GetFile** or **GetMultipleFiles** methods. This will be followed by one or more **OnProgress** events which will indicate the progress of the transfer. If multiple files are being downloaded, this event will fire for each file as it is transferred.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.GetFileEventArgs Class

Provides data for the OnGetFile event.

For a list of all members of this type, see FtpClient.GetFileEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FtpClient.GetFileEventArgs

[Visual Basic]
```
Public Class FtpClient.GetFileEventArgs
    Inherits EventArgs
```

[C#]
```
public class FtpClient.GetFileEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

GetFileEventArgs specifies information about the start of a file transfer from the server to the local system.

The OnGetFile event occurs when either the **GetFile** or **GetMultipleFiles** methods are called.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

FtpClient.GetFileEventArgs Members | SocketTools Namespace

FtpClient.GetFileEventArgs Members

FtpClient.GetFileEventArgs overview

Public Instance Constructors

FtpClient.GetFileEventArgs Constructor	Initializes a new instance of the FtpClient.GetFileEventArgs class.

Public Instance Properties

LocalFile	Gets a value which specifies the name of the local file.
RemoteFile	Gets a value which specifies the name of the remote file.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpClient.GetFileEventArgs Class | SocketTools Namespace

FtpClient.GetFileEventArgs Constructor

Initializes a new instance of the FtpClient.GetFileEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FtpClient.GetFileEventArgs();
```

See Also

FtpClient.GetFileEventArgs Class | SocketTools Namespace

FtpClient.GetFileEventArgs Properties

The properties of the **FtpClient.GetFileEventArgs** class are listed below. For a complete list of **FtpClient.GetFileEventArgs** class members, see the FtpClient.GetFileEventArgs Members topic.

Public Instance Properties

LocalFile	Gets a value which specifies the name of the local file.
RemoteFile	Gets a value which specifies the name of the remote file.

See Also

FtpClient.GetFileEventArgs Class | SocketTools Namespace

FtpClient.GetFileEventArgs.LocalFile Property

Gets a value which specifies the name of the local file.

[Visual Basic]

```
Public ReadOnly Property LocalFile As String
```

[C#]

```
public string LocalFile {get;}
```

Property Value

A string which specifies the name of the local file.

See Also

FtpClient.GetFileEventArgs Class | SocketTools Namespace

FtpClient.GetFileEventArgs.RemoteFile Property

Gets a value which specifies the name of the remote file.

[Visual Basic]

```
Public ReadOnly Property RemoteFile As String
```

[C#]

```
public string RemoteFile {get;}
```

Property Value

A string value which specifies the name of the remote file.

See Also

FtpClient.GetFileEventArgs Class | SocketTools Namespace

FtpClient.OnProgress Event

Occurs as a data stream is being read or written to the socket.

[Visual Basic]
```
Public Event OnProgress As OnProgressEventHandler
```

[C#]
```
public event OnProgressEventHandler OnProgress;
```

Event Data

The event handler receives an argument of type FtpClient.ProgressEventArgs containing data related to this event. The following **FtpClient.ProgressEventArgs** properties provide information specific to this event.

Property	Description
BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
BytesTotal	Gets a value which specifies the total number of bytes in the data stream.
FileName	Gets a value which specifies a file name.
Percent	Gets a value which specifies the percentage of data that has been read or written.

Remarks

The **OnProgress** event occurs as a data stream is being read or written to the socket. If large amounts of data are being read or written, this event can be used to update a progress bar or other user-interface component to provide the user with some visual feedback on the progress of the operation.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.ProgressEventArgs Class

Provides data for the OnProgress event.

For a list of all members of this type, see FtpClient.ProgressEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FtpClient.ProgressEventArgs

[Visual Basic]

```
Public Class FtpClient.ProgressEventArgs
    Inherits EventArgs
```

[C#]

```
public class FtpClient.ProgressEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ProgressEventArgs specifies the number of bytes copied from the data stream, the total number of bytes in the data stream and a completion percentage.

The OnProgress event occurs as a data stream is being read or written to the socket.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

FtpClient.ProgressEventArgs Members | SocketTools Namespace

FtpClient.ProgressEventArgs Members

FtpClient.ProgressEventArgs overview

Public Instance Constructors

FtpClient.ProgressEventArgs Constructor	Initializes a new instance of the FtpClient.ProgressEventArgs class.

Public Instance Properties

BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
BytesTotal	Gets a value which specifies the total number of bytes in the data stream.
FileName	Gets a value which specifies a file name.
Percent	Gets a value which specifies the percentage of data that has been read or written.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpClient.ProgressEventArgs Class | SocketTools Namespace

FtpClient.ProgressEventArgs Constructor

Initializes a new instance of the FtpClient.ProgressEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FtpClient.ProgressEventArgs();
```

See Also

FtpClient.ProgressEventArgs Class | SocketTools Namespace

FtpClient.ProgressEventArgs Properties

The properties of the **FtpClient.ProgressEventArgs** class are listed below. For a complete list of **FtpClient.ProgressEventArgs** class members, see the FtpClient.ProgressEventArgs Members topic.

Public Instance Properties

BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
BytesTotal	Gets a value which specifies the total number of bytes in the data stream.
FileName	Gets a value which specifies a file name.
Percent	Gets a value which specifies the percentage of data that has been read or written.

See Also

FtpClient.ProgressEventArgs Class | SocketTools Namespace

FtpClient.ProgressEventArgs.BytesCopied Property

Gets a value which specifies the number of bytes of data that has been read or written.

[Visual Basic]

```
Public ReadOnly Property BytesCopied As Long
```

[C#]

```
public long BytesCopied {get;}
```

Property Value

An integer value which specifies the number of bytes of data.

Remarks

The **BytesCopied** property specifies the number of bytes that have been read from the socket and stored in the local stream buffer, or written from the stream buffer to the socket.

See Also

FtpClient.ProgressEventArgs Class | SocketTools Namespace | BytesTotal Property | Percent Property

FtpClient.ProgressEventArgs.BytesTotal Property

Gets a value which specifies the total number of bytes in the data stream.

[Visual Basic]

```
Public ReadOnly Property BytesTotal As Long
```

[C#]

```
public long BytesTotal {get;}
```

Property Value

An integer value which specifies the number of bytes of data.

Remarks

The **BytesTotal** property specifies the total amount of data being read from the socket and stored in the data stream, or written from the data stream to the socket. If the amount of data was unknown or unspecified at the time the method call was made, then this value will always be the same as the **BytesCopied** property.

See Also

FtpClient.ProgressEventArgs Class | SocketTools Namespace | BytesCopied Property | Percent Property

FtpClient.ProgressEventArgs.FileName Property

Gets a value which specifies a file name.

[Visual Basic]

```
Public ReadOnly Property FileName As String
```

[C#]

```
public string FileName {get;}
```

Property Value

A string value which specifies the name of the file being transferred.

See Also

FtpClient.ProgressEventArgs Class | SocketTools Namespace

FtpClient.ProgressEventArgs.Percent Property

Gets a value which specifies the percentage of data that has been read or written.

[Visual Basic]

```
Public ReadOnly Property Percent As Integer
```

[C#]

```
public int Percent {get;}
```

Property Value

An integer value which specifies a percentage.

Remarks

The **Percent** property specifies the percentage of data that has been transmitted, expressed as an integer value between 0 and 100, inclusive. If the maximum size of the data stream was not specified by the caller, this value will always be 100.

See Also

FtpClient.ProgressEventArgs Class | SocketTools Namespace | BytesCopied Property | BytesTotal Property

FtpClient.OnPutFile Event

Occurs when a file upload is initiated.

[Visual Basic]
```
Public Event OnPutFile As OnPutFileEventHandler
```

[C#]
```
public event OnPutFileEventHandler OnPutFile;
```

Event Data

The event handler receives an argument of type FtpClient.PutFileEventArgs containing data related to this event. The following **FtpClient.PutFileEventArgs** properties provide information specific to this event.

Property	Description
LocalFile	Gets a value which specifies the name of the local file.
RemoteFile	Gets a value which specifies the name of the remote file.

Remarks

The **OnPutFile** event is generated when a file transfer is initiated by calling the **PutFile** or **PutMultipleFiles** methods. This will be followed by one or more **OnProgress** events which will indicate the progress of the transfer. If multiple files are being uploaded, this event will fire for each file as it is transferred.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.PutFileEventArgs Class

Provides data for the OnPutFile event.

For a list of all members of this type, see FtpClient.PutFileEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FtpClient.PutFileEventArgs

[Visual Basic]

```
Public Class FtpClient.PutFileEventArgs
    Inherits EventArgs
```

[C#]

```
public class FtpClient.PutFileEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

PutFileEventArgs specifies information about the start of a file transfer from the local system to the server.

The OnPutFile event occurs when either the **PutFile** or **PutMultipleFiles** methods are called.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

FtpClient.PutFileEventArgs Members | SocketTools Namespace

FtpClient.PutFileEventArgs Members

FtpClient.PutFileEventArgs overview

Public Instance Constructors

FtpClient.PutFileEventArgs Constructor	Initializes a new instance of the FtpClient.PutFileEventArgs class.

Public Instance Properties

LocalFile	Gets a value which specifies the name of the local file.
RemoteFile	Gets a value which specifies the name of the remote file.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpClient.PutFileEventArgs Class | SocketTools Namespace

FtpClient.PutFileEventArgs Constructor

Initializes a new instance of the FtpClient.PutFileEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public FtpClient.PutFileEventArgs();
```

See Also

FtpClient.PutFileEventArgs Class | SocketTools Namespace

FtpClient.PutFileEventArgs Properties

The properties of the **FtpClient.PutFileEventArgs** class are listed below. For a complete list of **FtpClient.PutFileEventArgs** class members, see the FtpClient.PutFileEventArgs Members topic.

Public Instance Properties

LocalFile	Gets a value which specifies the name of the local file.
RemoteFile	Gets a value which specifies the name of the remote file.

See Also

FtpClient.PutFileEventArgs Class | SocketTools Namespace

FtpClient.PutFileEventArgs.LocalFile Property

Gets a value which specifies the name of the local file.

[Visual Basic]
```
Public ReadOnly Property LocalFile As String
```

[C#]
```
public string LocalFile {get;}
```

Property Value

A string which specifies the name of the local file.

See Also

FtpClient.PutFileEventArgs Class | SocketTools Namespace

FtpClient.PutFileEventArgs.RemoteFile Property

Gets a value which specifies the name of the remote file.

[Visual Basic]

```
Public ReadOnly Property RemoteFile As String
```

[C#]

```
public string RemoteFile {get;}
```

Property Value

A string value which specifies the name of the remote file.

See Also

FtpClient.PutFileEventArgs Class | SocketTools Namespace

FtpClient.OnRead Event

Occurs when data is available to be read from the socket.

[Visual Basic]

```
Public Event OnRead As EventHandler
```

[C#]

```
public event EventHandler OnRead;
```

Remarks

The **OnRead** event occurs when data is available to be read from the server. This event is level-triggered, which means that once this event fires, it will not occur again until some data has been read from the server. This design prevents an application from being flooded with event notifications. It is recommended that your application read all of the available data from the server and store it in a local buffer for processing. See the example below.

This event is only generated if the client is in non-blocking mode.

Example

```
Private Sub FtpClient1_OnRead(ByVal sender As Object, ByVal e As System.EventArgs)
Handles FtpClient1.OnRead
    Dim strBuffer As String
    Dim nRead As Integer

    Do
        ' Read up to m_nBufferSize bytes of data from the socket
        nRead = FtpClient1.Read(strBuffer, m_nBufferSize)

        If nRead > 0 Then
            ' Append the data to an internal buffer for processing
            m_dataBuffer = m_dataBuffer + strBuffer
        End If
    Loop Until nRead < 1

    ProcessData()
End Sub
```

See Also

FtpClient Class | SocketTools Namespace

FtpClient.OnTaskBegin Event

Occurs when an asynchronous task begins execution.

[Visual Basic]
```
Public Event OnTaskBegin As OnTaskBeginEventHandler
```

[C#]
```
public event OnTaskBeginEventHandler OnTaskBegin;
```

Event Data

The event handler receives an argument of type FtpClient.TaskBeginEventArgs containing data related to this event. The following **FtpClient.TaskBeginEventArgs** property provides information specific to this event.

Property	Description
TaskId	Get the unique task identifier associated with the event.

Remarks

The **OnTaskBegin** event occurs when a background task associated with an asynchronous file transfer begins executing. This event can be used in conjunction with the **OnTaskEnd** event to monitor one or more background tasks that are created to perform asynchronous file transfers.

This event and the related asynchronous task events are invoked from the context of the thread that is managing the background task, and not the thread that created the class instance. If a handler is implemented for this event, its code will be executing in a different thread than the main UI thread. You should never attempt to update your application's user interface directly from within this event handler. Instead, you must create a delegate and use the **Invoke** method to ensure that any changes to the user interface are done within the context of the main UI thread.

Because background tasks are managed in separate threads, this has the effect of making your application multi-threaded, even if you do not explicitly create any worker threads in your own code. If the code in your event handler modifies a public member variable or shared object, you must ensure that access to that object is synchronized. For example, if your event handler updates a shared instance of a **Hashtable** object, you should ensure that all operations are performed through the thread-safe wrapper returned by the **Synchronized** method for that class. Refer to the MSDN documentation for more information about creating thread-safe applications.

See Also

FtpClient Class | SocketTools Namespace | OnTaskEnd Event | OnTaskRun Event

FtpClient.TaskBeginEventArgs Class

Provides data for the OnTaskBegin event.

For a list of all members of this type, see FtpClient.TaskBeginEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FtpClient.TaskBeginEventArgs

[Visual Basic]

```
Public Class FtpClient.TaskBeginEventArgs
    Inherits EventArgs
```

[C#]

```
public class FtpClient.TaskBeginEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

FtpClient.TaskBeginEventArgs Members | SocketTools Namespace

FtpClient.TaskBeginEventArgs Constructor

Initializes a new instance of the FtpClient.TaskBeginEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FtpClient.TaskBeginEventArgs();
```

See Also

FtpClient.TaskBeginEventArgs Class | SocketTools Namespace

FtpClient.TaskBeginEventArgs Members

FtpClient.TaskBeginEventArgs overview

Public Instance Constructors

FtpClient.TaskBeginEventArgs Constructor	Initializes a new instance of the FtpClient.TaskBeginEventArgs class.

Public Instance Properties

TaskId	Get the unique task identifier associated with the event.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpClient.TaskBeginEventArgs Class | SocketTools Namespace

FtpClient.TaskBeginEventArgs Properties

The properties of the **FtpClient.TaskBeginEventArgs** class are listed below. For a complete list of **FtpClient.TaskBeginEventArgs** class members, see the FtpClient.TaskBeginEventArgs Members topic.

Public Instance Properties

TaskId	Get the unique task identifier associated with the event.

See Also

FtpClient.TaskBeginEventArgs Class | SocketTools Namespace

FtpClient.TaskBeginEventArgs.TaskId Property

Get the unique task identifier associated with the event.

[Visual Basic]

```
Public ReadOnly Property TaskId As Integer
```

[C#]

```
public int TaskId {get;}
```

Property Value

An integer value that uniquely identifies the task that invoked the event handler.

See Also

FtpClient.TaskBeginEventArgs Class | SocketTools Namespace

FtpClient.OnTaskEnd Event

Occurs when an asynchronous task completes.

[Visual Basic]

```
Public Event OnTaskEnd As OnTaskEndEventHandler
```

[C#]

```
public event OnTaskEndEventHandler OnTaskEnd;
```

Event Data

The event handler receives an argument of type FtpClient.TaskEndEventArgs containing data related to this event. The following **FtpClient.TaskEndEventArgs** properties provide information specific to this event.

Property	Description
Error	Get the last error code for the background task.
TaskId	Get the unique task identifier associated with the event.
TimeElapsed	Gets the amount of time that has elapsed in milliseconds.

Remarks

The **OnTaskEnd** event occurs when a file transfer completes and the background task has terminated. Refer to the **OnTaskBegin** event for additional information about implementing a handler for this event.

See Also

FtpClient Class | SocketTools Namespace | OnTaskBegin Event | OnTaskRun Event

FtpClient.TaskEndEventArgs Class

Provides data for the OnTaskEnd event.

For a list of all members of this type, see FtpClient.TaskEndEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FtpClient.TaskEndEventArgs

[Visual Basic]

```
Public Class FtpClient.TaskEndEventArgs
    Inherits EventArgs
```

[C#]

```
public class FtpClient.TaskEndEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

FtpClient.TaskEndEventArgs Members | SocketTools Namespace

FtpClient.TaskEndEventArgs Constructor

Initializes a new instance of the FtpClient.TaskEndEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public FtpClient.TaskEndEventArgs();
```

See Also

FtpClient.TaskEndEventArgs Class | SocketTools Namespace

FtpClient.TaskEndEventArgs Members

FtpClient.TaskEndEventArgs overview

Public Instance Constructors

FtpClient.TaskEndEventArgs Constructor	Initializes a new instance of the FtpClient.TaskEndEventArgs class.

Public Instance Properties

Error	Get the last error code for the background task.
TaskId	Get the unique task identifier associated with the event.
TimeElapsed	Gets the amount of time that has elapsed in milliseconds.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpClient.TaskEndEventArgs Class | SocketTools Namespace

FtpClient.TaskEndEventArgs Properties

The properties of the **FtpClient.TaskEndEventArgs** class are listed below. For a complete list of **FtpClient.TaskEndEventArgs** class members, see the FtpClient.TaskEndEventArgs Members topic.

Public Instance Properties

Error	Get the last error code for the background task.
TaskId	Get the unique task identifier associated with the event.
TimeElapsed	Gets the amount of time that has elapsed in milliseconds.

See Also

FtpClient.TaskEndEventArgs Class | SocketTools Namespace

FtpClient.TaskEndEventArgs.Error Property

Get the last error code for the background task.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public FtpClient.ErrorCode Error {get;}
```

Property Value

An **ErrorCode** enumeration that specifies the last error code set by the background task.

See Also

FtpClient.TaskEndEventArgs Class | SocketTools Namespace

FtpClient.TaskEndEventArgs.TaskId Property

Get the unique task identifier associated with the event.

[Visual Basic]

```
Public ReadOnly Property TaskId As Integer
```

[C#]

```
public int TaskId {get;}
```

Property Value

An integer value that uniquely identifies the task that invoked the event handler.

See Also

FtpClient.TaskEndEventArgs Class | SocketTools Namespace

FtpClient.TaskEndEventArgs.TimeElapsed Property

Gets the amount of time that has elapsed in milliseconds.

[Visual Basic]

```
Public ReadOnly Property TimeElapsed As Integer
```

[C#]

```
public int TimeElapsed {get;}
```

Property Value

An integer value that specifies the number of milliseconds that the background task has executed.

See Also

FtpClient.TaskEndEventArgs Class | SocketTools Namespace

FtpClient.OnTaskRun Event

Occurs while a background task is active.

[Visual Basic]
```
Public Event OnTaskRun As OnTaskRunEventHandler
```

[C#]
```
public event OnTaskRunEventHandler OnTaskRun;
```

Event Data

The event handler receives an argument of type FtpClient.TaskRunEventArgs containing data related to this event. The following **FtpClient.TaskRunEventArgs** properties provide information specific to this event.

Property	Description
Completed	Gets an estimate of the progress of the background task.
TaskId	Get the unique task identifier associated with the event.
TimeElapsed	Gets the amount of time that has elapsed in milliseconds.

Remarks

The **OnTaskRun** event is generated periodically during a file transfer while the background task is active. The rate and number of times that this event will be generated depends on the task being performed. This event is generally analogous to the **OnProgress** event for file transfers that are performed in the current working thread, however the **OnTaskRun** event will occur for each individual background task that is active.

Refer to the **OnTaskBegin** event for additional information about implementing a handler for this event.

See Also

FtpClient Class | SocketTools Namespace | OnTaskBegin Event | OnTaskEnd Event

FtpClient.TaskRunEventArgs Class

Provides data for the OnTaskRun event.

For a list of all members of this type, see FtpClient.TaskRunEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FtpClient.TaskRunEventArgs

[Visual Basic]

```
Public Class FtpClient.TaskRunEventArgs
    Inherits EventArgs
```

[C#]

```
public class FtpClient.TaskRunEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

FtpClient.TaskRunEventArgs Members | SocketTools Namespace

FtpClient.TaskRunEventArgs Constructor

Initializes a new instance of the FtpClient.TaskRunEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public FtpClient.TaskRunEventArgs();
```

See Also

FtpClient.TaskRunEventArgs Class | SocketTools Namespace

FtpClient.TaskRunEventArgs Members

FtpClient.TaskRunEventArgs overview

Public Instance Constructors

FtpClient.TaskRunEventArgs Constructor	Initializes a new instance of the FtpClient.TaskRunEventArgs class.

Public Instance Properties

Completed	Gets an estimate of the progress of the background task.
TaskId	Get the unique task identifier associated with the event.
TimeElapsed	Gets the amount of time that has elapsed in milliseconds.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpClient.TaskRunEventArgs Class | SocketTools Namespace

FtpClient.TaskRunEventArgs Properties

The properties of the **FtpClient.TaskRunEventArgs** class are listed below. For a complete list of **FtpClient.TaskRunEventArgs** class members, see the FtpClient.TaskRunEventArgs Members topic.

Public Instance Properties

Completed	Gets an estimate of the progress of the background task.
TaskId	Get the unique task identifier associated with the event.
TimeElapsed	Gets the amount of time that has elapsed in milliseconds.

See Also

FtpClient.TaskRunEventArgs Class | SocketTools Namespace

FtpClient.TaskRunEventArgs.Completed Property

Gets an estimate of the progress of the background task.

[Visual Basic]

```
Public ReadOnly Property Completed As Integer
```

[C#]

```
public int Completed {get;}
```

Property Value

An integer value that returns a number between 0 and 100 inclusive that specifies the estimated percentage of completion for the task. A value of zero indicates that the task has just begun executing, while a value of 100 indicates that the task is at or near completion.

See Also

FtpClient.TaskRunEventArgs Class | SocketTools Namespace

FtpClient.TaskRunEventArgs.TaskId Property

Get the unique task identifier associated with the event.

[Visual Basic]

```
Public ReadOnly Property TaskId As Integer
```

[C#]

```
public int TaskId {get;}
```

Property Value

An integer value that uniquely identifies the task that invoked the event handler.

See Also

FtpClient.TaskRunEventArgs Class | SocketTools Namespace

FtpClient.TaskRunEventArgs.TimeElapsed Property

Gets the amount of time that has elapsed in milliseconds.

[Visual Basic]
```
Public ReadOnly Property TimeElapsed As Integer
```

[C#]
```
public int TimeElapsed {get;}
```

Property Value

An integer value that specifies the number of milliseconds that the background task has been executing.

See Also

FtpClient.TaskRunEventArgs Class | SocketTools Namespace

FtpClient.OnTimeout Event

Occurs when a blocking operation fails to complete before the timeout period elapses.

[Visual Basic]

```
Public Event OnTimeout As EventHandler
```

[C#]

```
public event EventHandler OnTimeout;
```

Remarks

The **OnTimeout** event occurs when a blocking operation, such as sending or receiving data on the socket, fails to complete before the specified timeout period elapses. The timeout period for a blocking operation can be adjusted by setting the **Timeout** property.

This event is only generated if the client is in blocking mode.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.OnWrite Event

Occurs when data can be written to the socket.

[Visual Basic]

```
Public Event OnWrite As EventHandler
```

[C#]

```
public event EventHandler OnWrite;
```

Remarks

The **OnWrite** event occurs when the application can write data to the server. This event will typically occur when a connection is first established with the remote host, and after the **Write** method has failed because there was insufficient memory available in the socket send buffers. In the second case, when some of the buffered data has been successfully sent to the remote host and there is space available in the send buffers, this event is used to signal the application that it may attempt to send more data.

This event is only generated if the client is in non-blocking mode.

See Also

FtpClient Class | SocketTools Namespace

FtpClient.ErrorCode Enumeration

Specifies the error codes returned by the FtpClient class.

[Visual Basic]
```
Public Enum FtpClient.ErrorCode
```

[C#]
```
public enum FtpClient.ErrorCode
```

Remarks

The FtpClient class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.
errorProxyAuthenticationRequired	Proxy authentication required.

errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.
errorUnsupportedMediaType	Request specified an unsupported media type.

errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.
errorNoMoreMailboxes	No more mailboxes exist on this server.

errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.
errorInvalidThreadHandle	The specified thread handle is no longer valid.

errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been defined.

errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or unrecognized.

errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

FtpClient.FtpChannelMode Enumeration

Specifies the channel mode specified by setting the ChannelMode property.

[Visual Basic]

```
Public Enum FtpClient.FtpChannelMode
```

[C#]

```
public enum FtpClient.FtpChannelMode
```

Members

Member Name	Description
channelClear	Data sent and received on this channel should not be encrypted.
channelSecure	Data sent and received on this channel should be encrypted. Specifying this option requires that a secure connection has already been established with the server.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.FtpChannelType Enumeration

Specifies the channel used by the ChannelMode property.

[Visual Basic]

```
Public Enum FtpClient.FtpChannelType
```

[C#]

```
public enum FtpClient.FtpChannelType
```

Members

Member Name	Description
channelCommand	The communication channel used to send commands to the server and receive command result and status information from the server.
channelData	The communication channel used to send or receive data during a file transfer.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.FtpDirectoryFormat Enumeration

A value which specifies the format of a directory listing returned by the server.

[Visual Basic]
```
Public Enum FtpClient.FtpDirectoryFormat
```

[C#]
```
public enum FtpClient.FtpDirectoryFormat
```

Remarks

Values other than **formatAuto** should only be set if the class library cannot automatically determine the directory format returned by the server. The default directory format is determined both by the server's operating system and by analyzing the format of the data returned by the server. If the class is unable to automatically determine the format, it will attempt to parse the list of files as though it is a UNIX style listing.

Members

Member Name	Description
formatAuto	This value specifies that the control should automatically determine the format of the file lists returned by the server. It is recommended that most applications use this value and allow the control to automatically determine the appropriate file listing format used by the server.
formatUnix	This value specifies that the server returns file lists in the format commonly used by UNIX servers. Note that many servers can be configured to return file listings in this format, even if they are not actually a UNIX based platform. Consult the technical reference documentation for your server for more information.
formatMsdos	This value specifies that the server returns file lists in the format commonly used by MS-DOS based systems. This includes Windows NT servers. Long file names will be returned if supported by the underlying filesystem, such as NTFS or FAT32.
formatVms	This value specifies that the server returns file lists in the format commonly used by VMS servers. Note that VMS servers can be configured to return a standard UNIX style listing in additional to the default VMS format.
formatSterling1	This value specifies that the server returns file listings in a proprietary format used by the Sterling server, which is used for EDI (Electronic Data Interchange) applications. This format uses a 13 byte status code.
formatSterling2	This value specifies that the server returns file

	listings in a proprietary format used by the Sterling server, which is used for EDI (Electronic Data Interchange) applications. This format uses a 10 byte status code.
formatNetware	This value specifies that the server returns file listings in a proprietary format used by NetWare servers. The format is similar to UNIX style listings except that file access and permissions are indicated by letter codes enclosed in brackets. This is the default format selected if the server identifies itself as a NetWare system.
formatMlsd	This value specifies that the server should return file listings in a machine-independent format as defined by RFC 3659. This format specifies file information as a sequence of name and value pairs, with the same format being used regardless of the operating system that the server is hosted on. Note that not all servers support this format, and some proxy servers may reject the command even if the remote server supports its use.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.FtpFeatures Enumeration

Specifies the server features that are available for the current client session.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum FtpClient.FtpFeatures
```

```
[C#]
[Flags]
public enum FtpClient.FtpFeatures
```

Remarks

When a client connection is first established, all features are enabled by default. However, as the client issues commands to the server, if the server reports that the command is unrecognized that feature will automatically be disabled in the client.

For example, the first time an application calls the **GetFileSize** method to determine the size of a file, the class library will try to use the SIZE command. If the server reports that the SIZE command is not available, that feature will be disabled and the class will not use the command again during the session unless it is explicitly re-enabled. This is designed to prevent the class from repeatedly sending invalid commands to a server, which may result in the server aborting the connection.

Members

Member Name	Description	Value
featureSIZE	The server supports the SIZE command to determine the size of a file. If this feature is not enabled, the library will attempt to use the STAT command to determine the file size.	1
featureSTAT	The server supports using the STAT command to return information about a specific file. If this feature is not enabled, the client may not be able to obtain information about a specific file such as its size, permissions or modification time.	2
featureMDTM	The server supports the MDTM command to obtain information about the modification time for a specific file. This command may also be used to set the file time on the server.	4
featureREST	The server supports restarting file transfers using the REST command. If this feature is not enabled, the client will not be able to restart file transfers and must upload or download the complete file.	8

featureSITE	The server supports site specific commands using the SITE command. If this feature is not enabled, no site specific commands will be sent to the server.	16
featureIDLE	The server supports setting the idle timeout period using the SITE IDLE command to specify the number of seconds that the client may idle before the server terminates the connection.	32
featureCHMOD	The server supports modifying the permissions of a specific file using the SITE CHMOD command. If this feature is not enabled, the client will not be able to set the permissions for a file.	64
featureAUTH	The server supports explicit SSL sessions using the AUTH command. If this feature is not enabled, the client will only be able to connect to a secure server that uses implicit SSL connections. Changing this feature has no effect on standard, non-secure connections.	128
featurePBSZ	The server supports the PBSZ command which specifies the buffer size used with secure data connections. If this feature is disabled, it may prevent the client from changing the protection level on the data channel. Changing this feature has no effect on standard, non-secure connections.	256
featurePROT	The server supports the PROT command which specifies the protection level for the data channel. If this feature is disabled, the client will be unable to change the protection level on the data channel. Changing this feature has no effect on standard, non-secure connections.	512
featureCCC	The server supports the CCC command which returns the command channel to a non-secure mode. Changing this feature has no effect on standard, non-secure connections.	1024
featureHOST	The server supports the HOST command which enables a client to specify the hostname after establishing a connection with a server that supports virtual hosting.	2048

featureMLST	The server supports the MLST command which returns status information for files. If this feature is enabled, the MLST command will be used instead of the STAT command.	4096
featureMFMT	The server supports the MFMT command which is used to change the last modification time for a file. If this command is supported, it is used instead of the MDTM command to change the modification time for a file.	8192
featureXCRC	The server supports the XCRC command which returns the CRC32 checksum for the contents of a specified file. This command is used for file verification.	16384
featureXMD5	The server supports the XMD5 command which returns an MD5 hash for the contents of a specified file. This command is used for file verification.	32768
featureLANG	The server supports the LANG command which sets the language used for the current client session. Command responses and file naming conventions will use the specified language.	65536
featureUTF8	The server supports the OPTS UTF-8 command which specifies UTF-8 encoding when specifying filenames. This feature is typically used in conjunction with setting the default language for the client session.	131072
featureXQUOTA	The server supports the XQUOTA command which returns quota information for the current client session.	262144
featureUTIME	The server supports the UTIME command which is used to change the last modification time for a specified file.	524288

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.FtpFileType Enumeration

Specifies the type of file being transferred.

[Visual Basic]

```
Public Enum FtpClient.FtpFileType
```

[C#]

```
public enum FtpClient.FtpFileType
```

Members

Member Name	Description
fileAuto	The file type should be automatically determined based on the file name extension. If the file extension is unknown, the file type should be determined based on the contents of the file. The class has an internal list of common text file extensions, and additional file extensions can be registered using the **AddFileType** method.
fileAscii	The file being transferred is an ASCII text file. The characters the mark the end of a line (for example, a carriage return/linefeed pair under MS-DOS) are automatically converted to the format used by the target operating system.
fileEbcdic	The file being transferred is a text file created using the EBCDIC character set. If a file is being copied to a remote system, the ASCII characters are automatically converted to EBCDIC. If the file is being retrieved from a remote system, the EBCDIC characters are automatically converted to ASCII.
fileImage	The file is transferred without any modification. This is the default file transfer type, and should be used when transferring binary (non-text) data.
fileText	The same value as **fileAscii**.
fileBinary	The same value as **fileImage**.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace | AddFileType Method (SocketTools.FtpClient)

FtpClient.FtpOpenMode Enumeration

Specifies how a file is opened on the server.

[Visual Basic]

```
Public Enum FtpClient.FtpOpenMode
```

[C#]

```
public enum FtpClient.FtpOpenMode
```

Members

Member Name	Description
fileRead	The file is opened for read access. If the file does not exist, an error will occur.
fileWrite	The file is opened for write access. If the file does not exist, it will be created. If it does exist, it will be overwritten.
fileAppend	The file is opened for write access. If the file does not exist, it will be created. If it does exist, the data will be appended to the end of the file.
fileUnique	The file is opened for write access and created using a file name that is guaranteed to be unique. This option is not supported on all servers.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.FtpOptions Enumeration

Specifies the options that the FtpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum FtpClient.FtpOptions
```

[C#]

```
[Flags]
public enum FtpClient.FtpOptions
```

Remarks

The FtpClient class uses the **FtpOptions** enumeration to specify one or more options to be used when establishing a connection with a remote host. Multiple options may be specified if necessary.

Members

Member Name	Description	Value
optionNone	No option specified.	0
optionDefault	The default connection option. This is the same as specifying **optionPassive**.	1
optionPassive	This option specifies the client should attempt to establish a passive connection to the server. This means that instead of the client opening a port on the local system and waiting for the server to establish a connection back to the client, the client will establish a second data connection to the server. This mode is recommended for most systems that are behind a NAT router or firewall.	1
optionFirewall	This option specifies the client should always use the host IP address to establish the data connection with the server, not the address returned by the server in response to the PASV command. This option may be necessary if the server is behind a router that performs Network Address Translation (NAT) and it returns an unreachable IP address for the data connection. If this option is specified, it will also enable passive mode data transfers.	2
optionNoAuth	This option specifies the server does not require authentication, or that it	4

	requires an alternate authentication method. When this option is used, the client connection is flagged as authenticated as soon as the connection to the server has been established. Note that using this option to bypass authentication may result in subsequent errors when attempting to retrieve a directory listing or transfer a file. It is recommended that you consult the technical reference documentation for the server to determine its specific authentication requirements.	
optionKeepAlive	This option specifies the client should attempt to keep the connection with the server active for an extended period of time. It is important to note that regardless of this option, the server may still choose to disconnect client sessions that are holding the command channel open but are not performing file transfers.	8
optionNoAuthRSA	This option specifies that RSA authentication should not be used with SSH-1 connections. This option is ignored with SSH-2 connections and should only be specified if required by the remote host. This option has no effect on standard or secure connections using SSL.	16
optionNoPwdNul	This option specifies that the user password cannot be terminated with a null byte. This option is ignored with SSH-2 connections and should only be specified if required by the remote host. This option has no effect on standard or secure connections using SSL.	32
optionNoRekey	This option specifies the client should never attempt a repeat key exchange with the server. Some SSH servers do not support rekeying the session, and this can cause the client to become non-responsive or abort the connection after being connected for an hour. This option has no effect on standard or secure connections using SSL.	64
optionCompatSID	This compatibility option changes how the session ID is handled during public key authentication with older SSH servers. This option should only be	128

	specified when connecting to servers that use OpenSSH 2.2.0 or earlier versions. This option has no effect on standard or secure connections using SSL.	
optionCompatHMAC	This compatibility option changes how the HMAC authentication codes are generated. This option should only be specified when connecting to servers that use OpenSSH 2.2.0 or earlier versions. This option has no effect on standard or secure connections using SSL.	256
optionVirtualHost	This option specifies the server supports virtual hosting, where multiple domains are hosted by a server using the same external IP address. If this option is enabled, the client will send the HOST command to the server upon establishing a connection.	512
optionVerify	This option specifies that file transfers should be automatically verified after the transfer has completed. If the server supports the XMD5 command, the transfer will be verified by calculating an MD5 hash of the file contents. If the server does not support the XMD5 command, but does support the XCRC command, the transfer will be verified by calculating a CRC32 checksum of the file contents. If neither the XMD5 or XCRC commands are supported, the transfer is verified by comparing the size of the file. Automatic file verification is only performed for binary mode transfers because of the end-of-line conversion that may occur when text files are uploaded or downloaded.	1024
optionTunnel	This option specifies that a tunneled TCP connection and/or port-forwarding is being used to establish the connection to the server. This changes the behavior of the client with regards to internal checks of the destination IP address and remote port number, default capability selection and how the connection is established. This option also forces all connections to be outbound and enables the firewall compatibility features in the client.	65536

optionTrustedSite	This option specifies the server is trusted. The server certificate will not be validated and the connection will always be permitted. This option only affects connections using either the SSL or TLS protocols.	2048
optionSecure	This option specifies the client should attempt to establish a secure connection with the server. The server must support secure connections using either the SSL or TLS protocol.	4096
optionImplicitSSL	This option specifies the client should attempt to establish a secure implicit SSL session. The SSL handshake is initiated immediately after the connection to the server has been established.	4096
optionExplicitSSL	This option specifies the client should attempt to establish a secure explicit SSL session. The initial connection to the server is not encrypted, and the client will attempt to negotiate a secure connection by sending a command to the server. Some servers may require this option when connecting to the server on ports other than the default secure port of 990.	8192
optionSecureShell	This option specifies the client should attempt to establish a secure connection using the Secure Shell (SSH) protocol. This option is automatically selected if the connection is established on port 22, the standard port for SSH connections. It is only necessary to specify this option if the SSH connection must be established on a non-standard port.	16384
optionSecureFallback	This option specifies the client should permit the use of less secure cipher suites for compatibility with legacy servers. If this option is specified, the client will permit connections using TLS 1.0 and cipher suites that use RC4, MD5 and SHA1.	32768
optionPreferIPv6	This option specifies the client should prefer the use of IPv6 if the server hostname can be resolved to both an IPv6 and IPv4 address. This option is ignored if the local system does not	262144

	have IPv6 enabled, or when the hostname can only be resolved to an IPv4 address. If the server hostname can only be resolved to an IPv6 address, the client will attempt to establish a connection using IPv6 regardless if this option has been specified.	
optionFreeThread	This option specifies that class methods may be called from any thread, and not only the thread that established the connection. Using this option disables certain internal safety checks that are made by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads.	524288
optionHiResTimer	This option specifies that elapsed time values should be returned in milliseconds rather than seconds. This option is intended to provide greater accuracy with smaller file transfers over a high speed network connection.	1048576
optionTLSReuse	This option specifies that TLS session reuse should be enabled when establishing a secure data connection. This option is only supported on Windows 8.1 or Windows Server 2012 R2 and later platforms, and it should only be used when explicitly required by the server. This option is not compatible with servers built using OpenSSL 1.0.2 and earlier versions which do not provide Extended Master Secret (EMS) support as outlined in RFC7627.	2097152

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.FtpPermissions Enumeration

Specifies the access permissions for a file on the server.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum FtpClient.FtpPermissions
```

```
[C#]
[Flags]
public enum FtpClient.FtpPermissions
```

Members

Member Name	Description	Value
symbolicLink	The file is a symbolic link to another file. Symbolic links are special types of files found on UNIX based systems which are similar to Windows shortcuts.	4096
ownerRead	The owner has permission to open the file for reading. If the current user is the owner of the file, this grants the user the right to download the file to the local system.	1024
ownerWrite	The owner has permission to open the file for writing. If the current user is the owner of the file, this grants the user the right to replace the file. If this permission is set for a directory, this grants the user the right to create and delete files.	512
ownerExecute	The owner has permission to execute the contents of the file. The file is typically either a binary executable, script or batch file. If this permission is set for a directory, this may also grant the user the right to open that directory and search for files in that directory.	256
groupRead	Users in the specified group have permission to open the file for reading. If the current user is in the same group as the file owner, this grants the user the right to download the file.	64
groupWrite	Users in the specified group have permission to open the file for writing. On some platforms, this may also imply permission to delete the file. If the current user is in the same group as the	32

	file owner, this grants the user the right to replace the file. If this permission is set for a directory, this grants the user the right to create and delete files.	
groupExecute	Users in the specified group have permission to execute the contents of the file. If this permission is set for a directory, this may also grant the user the right to open that directory and search for files in that directory.	16
worldRead	All users have permission to open the file for reading. This permission grants any user the right to download the file to the local system.	4
worldWrite	All users have permission to open the file for writing. This permission grants any user the right to replace the file. If this permission is set for a directory, this grants any user the right to create and delete files.	2
worldExecute	All users have permission to execute the contents of the file. If this permission is set for a directory, this may also grant all users the right to open that directory and search for files in that directory.	1

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.FtpProxyType Enumeration

Specifies the type of proxy server that is being used by the FtpClient class.

[Visual Basic]
```
Public Enum FtpClient.FtpProxyType
```

[C#]
```
public enum FtpClient.FtpProxyType
```

Members

Member Name	Description
proxyNone	No proxy server is being used. This is the default value.
proxyUser	The client is not logged into the proxy server. The USER command is sent in the format username@ftpsite followed by the password. This is the format used with the Gauntlet proxy server.
proxyLogin	The client is logged into the proxy server. The USER command is then sent in the format username@ftpsite followed by the password. This is the format used by the InterLock proxy server.
proxyOpen	The client is not logged into the proxy server. The OPEN command is sent specifying the host name, followed by the username and password.
proxySite	The client is logged into the server. The SITE command is sent, specifying the host name, followed by the username and the password.
proxyOther	This special proxy type specifies that another, undefined proxy server is being used. The client connects to the proxy host, but does not attempt to authenticate the client. The application is responsible for negotiating with the proxy server, typically using the Command property to send specific command sequences.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.FtpServerType Enumeration

Specifies the type of server that the client is connected to.

[Visual Basic]
```
Public Enum FtpClient.FtpServerType
```

[C#]
```
public enum FtpClient.FtpServerType
```

Members

Member Name	Description
serverUnknown	The server type could not be determined by issuing the SYST command. The server may not support the command, or the command may only allowed when issued by an authenticated user.
serverUNIX	The server is running on a UNIX based operating system. This can include Linux and other variants, as well as operating systems which emulate UNIX style file pathing and directory listings.
serverMSDOS	The server is running on an MS-DOS based operating system. The server expects file pathing and naming conventions according to the standard MS-DOS format and returns directory listings similar to the output of the DIR command.
serverWindows	The server is running on a Windows based operating system. The server expects file pathing and naming conventions according to the standard Windows long filename format, and returns directory listings similar to the output of the DIR command. Note that Windows servers may be configured to return file and directory information in a format similar to UNIX systems, in which case the system may be identified as UNIX even though it is actually running on a Windows platform.
serverVMS	The server is running on a DEC VMS based operating system. The server expects file pathing and naming conventions specific to that operating system. Note that VMS servers may be configured to return file and directory information in a format similar to UNIX systems, in which case the system may be identified as UNIX even though it is actually running on a VMS platform.
serverNetware	The server is running on a NetWare based operating system. The server expects file pathing and naming conventions similar to the standard Windows long filename format, and returns

	directory listings that are similar to UNIX systems with the exception of the access and permissions flags for the file. Note that a NetWare system may return listings in different formats based on the filesystem and site specific options specified.
serverOther	The server type was not recognized. An attempt will be made to automatically determine the correct file pathing and naming conventions used by the server. To obtain a list of files on the server, it may be necessary to explicitly set the DirectoryFormat property to specify the directory listing format.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.FtpStatus Enumeration

Specifies the status values that may be returned by the FtpClient class.

[Visual Basic]

```
Public Enum FtpClient.FtpStatus
```

[C#]

```
public enum FtpClient.FtpStatus
```

Members

Member Name	Description
statusUnused	A client connection has not been established. Attempts to perform any network operations, such as sending or receiving data, will generate an error.
statusIdle	A client connection has been created, but is not currently in use. A blocking socket operation can be executed at this point.
statusConnect	The client is in the process of establishing a connection with a remote host.
statusRead	The client is in the process of receiving data from a remote host.
statusWrite	The client is in the process of sending data to a remote host.
statusDisconnect	The client connection is being closed and subsequent attempts to access that session will result in an error.
statusOpenFile	A file on the remote host is being opened.
statusCloseFile	A file on the remote host is being closed.
statusGetFile	A file is being downloaded from the remote host to the local system. No other blocking operation may be performed in the current thread while the file transfer is in progress.
statusPutFile	A file is being uploaded from the local system to the remote host. No other blocking operation may be performed in the current thread while the file transfer is in progress.
statusFileList	A file listing is being returned from the remote host. No other blocking operation may be performed in the current thread while the file listing is in progress, including downloading or uploading files.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.FtpTransferOptions Enumeration

Specifies the file transfer options that the FtpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum FtpClient.FtpTransferOptions
```

[C#]

```
[Flags]
public enum FtpClient.FtpTransferOptions
```

Members

Member Name	Description	Value
transferDefault	This option specifies the default transfer mode should be used. If the file exists, it will be overwritten.	0
transferAppend	This option specifies that if the file exists, data will be appended to the end of the file. If the file does not exist, it will be created.	1

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.SecurityCertificate Enumeration

Specifies the security certificate status values that may be returned by the FtpClient class.

[Visual Basic]
```
Public Enum FtpClient.SecurityCertificate
```

[C#]
```
public enum FtpClient.SecurityCertificate
```

Remarks

The FtpClient class uses the **SecurityCertificate** enumeration to identify the current status of the certificate that was provided by the remote host when a secure connection was established.

Members

Member Name	Description
certificateNone	No certificate information is available. A secure connection was not established with the server.
certificateValid	The certificate is valid.
certificateNoMatch	The certificate is valid, however the domain name specified in the certificate does not match the domain name of the remote host. The application can examine the **CertificateSubject** property to determine the site the certificate was issued to.
certificateExpired	The certificate has expired and is no longer valid. The application can examine the **CertificateExpires** property to determine when the certificate expired.
certificateRevoked	The certificate has been revoked and is no longer valid. It is recommended that the application immediately terminate the connection if this status is returned.
certificateUntrusted	The certificate has not been issued by a trusted authority, or the certificate is not trusted on the local host. It is recommended that the application immediately terminate the connection if this status is returned.
certificateInvalid	The certificate is invalid. This typically indicates that the internal structure of the certificate is damaged. It is recommended that the application immediately terminate the connection if this status is returned.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.SecureCipherAlgorithm Enumeration

Specifies the encryption algorithms that the FtpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum FtpClient.SecureCipherAlgorithm
```

[C#]
```
[Flags]
public enum FtpClient.SecureCipherAlgorithm
```

Remarks

The FtpClient class uses the **SecureCipherAlgorithm** enumeration to identify which encryption algorithm was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
cipherNone	No cipher has been selected. A secure connection has not been established with the remote host.	0
cipherRC2	The RC2 block cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.	1
cipherRC4	The RC4 stream cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.	2
cipherRC5	The RC5 block cipher was selected. This is a variable key length cipher which supports keys up to 2040 bits, in 8-bit increments.	4
cipherDES	The DES (Data Encryption Standard) block cipher was selected. This is a fixed key length cipher using 56-bit keys.	8
cipherDES3	The Triple DES block cipher was selected. This cipher encrypts the data three times using different keys, effectively using a 168-bit key length.	16
cipherDESX	A variant of the DES block cipher which XORs an extra 64-bits of the key before and after the plaintext has been encrypted, increasing the key size to 184 bits.	32
cipherAES	The Advanced Encryption Standard	64

	cipher (also known as the Rijndael cipher) is a fixed block size cipher which use a key size of 128, 192 or 256 bits. This cipher is supported on Windows XP SP3 SP3 and later versions of the operating system.	
cipherSkipjack	The Skipjack block cipher was selected. This is a fixed key length cipher, using 80-bit keys.	128
cipherBlowfish	The Blowfish block cipher was selected. This is a variable key length cipher up to 448 bits, using a 64-bit block size.	256

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.SecureHashAlgorithm Enumeration

Specifies the hash algorithms that the FtpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum FtpClient.SecureHashAlgorithm
```

[C#]

```
[Flags]
public enum FtpClient.SecureHashAlgorithm
```

Remarks

The FtpClient class uses the **SecureHashAlgorithm** enumeration to identify the message digest (hash) algorithm that was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
hashNone	No hash algorithm has been selected. This is not a secure connection with the server.	0
hashMD5	The MD5 algorithm was selected. This algorithm produces a 128-bit message digest. This algorithm is no longer considered to be cryptographically secure.	1
hashSHA	The SHA-1 algorithm was selected. This algorithm produces a 160-bit message digest. This algorithm is no longer considered to be cryptographically secure.	2
hashSHA256	The SHA-256 algorithm was selected. This algorithm produces a 256-bit message digest.	4
hashSHA384	The SHA-384 algorithm was selected. This algorithm produces a 384-bit message digest.	8
hashSHA512	The SHA-512 algorithm was selected. This algorithm produces a 512-bit message digest.	16

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.SecureKeyAlgorithm Enumeration

Specifies the key exchange algorithms that the FtpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum FtpClient.SecureKeyAlgorithm
```

[C#]

```
[Flags]
public enum FtpClient.SecureKeyAlgorithm
```

Remarks

The FtpClient class uses the **SecureKeyAlgorithm** enumeration to identify the key exchange algorithm that was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
keyExchangeNone	No key exchange algorithm has been selected. This is not a secure connection with the server.	0
keyExchangeRSA	The RSA public key exchange algorithm has been selected.	1
keyExchangeKEA	The KEA public key exchange algorithm has been selected. This is an improved version of the Diffie-Hellman public key algorithm.	2
keyExchangeDH	The Diffie-Hellman public key exchange algorithm has been selected.	4
keyExchangeECDH	The Elliptic Curve Diffie-Hellman key exchange algorithm was selected. This is a variant of the Diffie-Hellman algorithm which uses elliptic curve cryptography. This key exchange algorithm is only supported on Windows XP SP3 SP3 and later versions of the operating system.	8

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.SecurityProtocols Enumeration

Specifies the security protocols that the FtpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum FtpClient.SecurityProtocols
```

[C#]
```
[Flags]
public enum FtpClient.SecurityProtocols
```

Remarks

The FtpClient class uses the **SecurityProtocols** enumeration to specify one or more security protocols to be used when establishing a connection with a remote host. Multiple protocols may be specified if necessary and the actual protocol used will be negotiated with the remote host. It is recommended that most applications use **protocolDefault** when creating a secure connection.

Members

Member Name	Description	Value
protocolNone	No security protocol will be used, a secure connection will not be established.	0
protocolSSL2	The SSL 2.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections.	1
protocolSSL3	The SSL 3.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections. In most cases, this protocol is only selected if TLS is not supported by the server.	2
protocolTLS10	The TLS 1.0 protocol has been selected. This version of the protocol is commonly used by older servers and is the only version of TLS supported on Windows platforms prior to Windows 7 SP1 and Windows Server 2008 R2.	4
protocolTLS11	The TLS 1.1 protocol has been selected. This version of TLS is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of the	8

	operating system.	
protocolTLS12	The TLS 1.2 protocol has been selected. This is the default version of the protocol and is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of Windows. It is recommended that you use this version of TLS.	16
protocolSSH1	The Secure Shell 1.0 protocol has been selected. This version of the protocol has been deprecated and is no longer widely used. It is not recommended that this version of the protocol be used to establish a connection.	256
protocolSSH2	The Secure Shell 2.0 protocol has been selected. This is the most commonly used version of the protocol. It is recommended that this version of the protocol be used unless the server explicitly requires the client to use an earlier version.	512
protocolSSL	Any version of the Secure Sockets Layer (SSL) protocol should be used. The actual protocol version used will be negotiated with the remote host.	3
protocolTLS	Any version of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with with preference for TLS 1.2.	28
protocolTLS1	Version 1.0, 1.1 or 1.2 of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with preference for TLS 1.2.	28
protocolSSH	Any version of the the Secure Shell (SSH) protocol should be used. The actual protocol version used will be negotiated with the remote host.	768
protocolDefault	The default selection of security protocols will be used when establishing a connection. The TLS 1.2, 1.1 and 1.0 protocols will be negotiated with the host, in that order of preference. This option will always request the latest version of the preferred security protocols and is the recommended	16

	value.	
protocolUnknown	An unknown or unsupported security protocol has been specified. This value indicates an error condition.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.TraceOptions Enumeration

Specifies the logging options that the FtpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum FtpClient.TraceOptions
```

[C#]

```
[Flags]
public enum FtpClient.TraceOptions
```

Remarks

The FtpClient class uses the **TraceOptions** enumeration to specify what kind of debugging information is written to the trace logfile. These options are only meaningful when trace logging is enabled by setting the **Trace** property to **true**.

Members

Member Name	Description	Value
traceDefault	The default trace logging option. This is the same as specifying the **traceInfo** option.	0
traceInfo	All network function calls are written to the trace file. This is the default value.	0
traceError	Only those network function calls which fail are recorded in the trace file.	1
traceWarning	Only those network function calls which fail, or return values which indicate a warning, are recorded in the trace file.	2
traceHexDump	All network function calls are written to the trace file, plus all the data that is sent or received is displayed, in both ASCII and hexadecimal format.	4
traceProcess	All function calls in the current process are logged, rather than only those functions in the current thread. This option is useful for multithreaded applications that are using worker threads.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.OnCommandEventHandler Delegate

Represents the method that will handle the OnCommand event.

[Visual Basic]

```
Public Delegate Sub FtpClient.OnCommandEventHandler( _
   ByVal sender As Object, _
   ByVal e As CommandEventArgs _
)
```

[C#]

```
public delegate void FtpClient.OnCommandEventHandler(
      object sender,
      CommandEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A CommandEventArgs object that contains the event data.

Remarks

When you create an **OnCommandEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnCommandEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub FtpClient.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void FtpClient.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ErrorEventArgs object that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.OnFileListEventHandler Delegate

Represents the method that will handle the OnFileList event.

[Visual Basic]

```
Public Delegate Sub FtpClient.OnFileListEventHandler( _
   ByVal sender As Object, _
   ByVal e As FileListEventArgs _
)
```

[C#]

```
public delegate void FtpClient.OnFileListEventHandler(
      object sender,
      FileListEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A FileListEventArgs object which contains the event data.

Remarks

When you create an **OnFileListEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnFileListEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.OnGetFileEventHandler Delegate

Represents the method that will handle the OnGetFile event.

[Visual Basic]

```
Public Delegate Sub FtpClient.OnGetFileEventHandler( _
   ByVal sender As Object, _
   ByVal e As GetFileEventArgs _
)
```

[C#]

```
public delegate void FtpClient.OnGetFileEventHandler(
      object sender,
      GetFileEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A GetFileEventArgs object which contains the event data.

Remarks

When you create an **OnGetFileEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnGetFileEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.OnProgressEventHandler Delegate

Represents the method that will handle the OnProgress event.

[Visual Basic]

```
Public Delegate Sub FtpClient.OnProgressEventHandler( _
   ByVal sender As Object, _
   ByVal e As ProgressEventArgs _
)
```

[C#]

```
public delegate void FtpClient.OnProgressEventHandler(
      object sender,
      ProgressEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A ProgressEventArgs object that contains the event data.

Remarks

When you create an **OnProgressEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnProgressEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.OnPutFileEventHandler Delegate

Represents the method that will handle the OnPutFile event.

[Visual Basic]

```
Public Delegate Sub FtpClient.OnPutFileEventHandler( _
   ByVal sender As Object, _
   ByVal e As PutFileEventArgs _
)
```

[C#]

```
public delegate void FtpClient.OnPutFileEventHandler(
      object sender,
      PutFileEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A PutFileEventArgs object that contains the event data.

Remarks

When you create an **OnPutFileEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnPutFileEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.OnTaskBeginEventHandler Delegate

Represents the method that will handle the OnTaskBegin event.

[Visual Basic]
```
Public Delegate Sub FtpClient.OnTaskBeginEventHandler( _
   ByVal sender As Object, _
   ByVal e As TaskBeginEventArgs _
)
```

[C#]
```
public delegate void FtpClient.OnTaskBeginEventHandler(
      object sender,
      TaskBeginEventArgs e
   );
```

Parameters

sender
The source of the event.

e
A TaskBeginEventArgs object that contains the event data.

Remarks

When you create an **OnTaskBeginEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnTaskBeginEventHandler** delegate declaration.

This event handler will be invoked in the context of the worker thread that is managing the background task, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.OnTaskEndEventHandler Delegate

Represents the method that will handle the OnTaskEnd event.

[Visual Basic]

```
Public Delegate Sub FtpClient.OnTaskEndEventHandler( _
   ByVal sender As Object, _
   ByVal e As TaskEndEventArgs _
)
```

[C#]

```
public delegate void FtpClient.OnTaskEndEventHandler(
      object sender,
      TaskEndEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A TaskEndEventArgs object that contains the event data.

Remarks

When you create an **OnTaskEndEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnTaskEndEventHandler** delegate declaration.

This event handler will be invoked in the context of the worker thread that is managing the background task, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.OnTaskRunEventHandler Delegate

Represents the method that will handle the OnTaskRun event.

[Visual Basic]

```
Public Delegate Sub FtpClient.OnTaskRunEventHandler( _
   ByVal sender As Object, _
   ByVal e As TaskRunEventArgs _
)
```

[C#]

```
public delegate void FtpClient.OnTaskRunEventHandler(
      object sender,
      TaskRunEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A TaskRunEventArgs object that contains the event data.

Remarks

When you create an **OnTaskRunEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnTaskRunEventHandler** delegate declaration.

This event handler will be invoked in the context of the worker thread that is managing the background task, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

SocketTools Namespace

FtpClient.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see FtpClient.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.FtpClient.RuntimeLicenseAttribute

[Visual Basic]
```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class FtpClient.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]
```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class FtpClient.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the FtpClient class.

Example

```
<Assembly: SocketTools.FtpClient.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.FtpClient.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

FtpClient.RuntimeLicenseAttribute Members | SocketTools Namespace

FtpClient.RuntimeLicenseAttribute Members

FtpClient.RuntimeLicenseAttribute overview

Public Instance Constructors

FtpClient.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpClient.RuntimeLicenseAttribute Class | SocketTools Namespace

FtpClient.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public FtpClient.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the FtpClient class.

See Also

FtpClient.RuntimeLicenseAttribute Class | SocketTools Namespace

FtpClient.RuntimeLicenseAttribute Properties

The properties of the **FtpClient.RuntimeLicenseAttribute** class are listed below. For a complete list of **FtpClient.RuntimeLicenseAttribute** class members, see the FtpClient.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

FtpClient.RuntimeLicenseAttribute Class | SocketTools Namespace

FtpClient.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

FtpClient.RuntimeLicenseAttribute Class | SocketTools Namespace

FtpClientException Class

The exception that is thrown when a client error occurs.

For a list of all members of this type, see FtpClientException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.FtpClientException

[Visual Basic]
```
Public Class FtpClientException
    Inherits ApplicationException
```

[C#]
```
public class FtpClientException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A FtpClientException is thrown by the FtpClient class when an error occurs.

The default constructor for the FtpClientException class sets the **ErrorCode** property to the last error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpClient (in SocketTools.FtpClient.dll)

See Also

FtpClientException Members | SocketTools Namespace

FtpClientException Members

FtpClientException overview

Public Instance Constructors

FtpClientException	Overloaded. Initializes a new instance of the FtpClientException class.

Public Instance Properties

HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpClientException Class | SocketTools Namespace

FtpClientException Constructor

Initializes a new instance of the FtpClientException class with the last client error code.

Overload List

Initializes a new instance of the FtpClientException class with the last client error code.

public FtpClientException();

Initializes a new instance of the FtpClientException class with a specified error number.

public FtpClientException(int);

Initializes a new instance of the FtpClientException class with a specified error message.

public FtpClientException(string);

Initializes a new instance of the FtpClientException class with a specified error message and a reference to the inner exception that is the cause of this exception.

public FtpClientException(string,Exception);

See Also

FtpClientException Class | SocketTools Namespace

FtpClientException Constructor ()

Initializes a new instance of the FtpClientException class with the last client error code.

[Visual Basic]

```
Overloads Public Sub New()
```

[C#]

```
public FtpClientException();
```

Remarks

The ctor constructor sets the **ErrorCode** property to the last client error that occurred. For more information about the errors that may occur, refer to the FtpClient.ErrorCode enumeration.

See Also

FtpClientException Class | SocketTools Namespace | FtpClientException Constructor Overload List

FtpClientException Constructor (String)

Initializes a new instance of the FtpClientException class with a specified error message.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]

```
public FtpClientException(
   string message
);
```

Parameters

message

The error message that explains the reason for the exception.

Remarks

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

FtpClientException Class | SocketTools Namespace | FtpClientException Constructor Overload List

FtpClientException Constructor (String, Exception)

Initializes a new instance of the FtpClientException class with a specified error message and a reference to the inner exception that is the cause of this exception.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal inner As Exception _
)
```

[C#]

```
public FtpClientException(
   string message,
   Exception inner
);
```

Parameters

message

The error message that explains the reason for the exception.

inner

The exception that is the cause of the current exception. If the ***innerException*** parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

FtpClientException Class | SocketTools Namespace | FtpClientException Constructor Overload List

FtpClientException Constructor (Int32)

Initializes a new instance of the FtpClientException class with a specified error number.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

[C#]

```
public FtpClientException(
   int code
);
```

Parameters

code

An integer value which specifies an error code.

Remarks

This constructor sets the **ErrorCode** property to the specified error code. For more information about the errors that may occur, refer to the FtpClient.ErrorCode enumeration.

See Also

FtpClientException Class | SocketTools Namespace | FtpClientException Constructor Overload List

FtpClientException Properties

The properties of the **FtpClientException** class are listed below. For a complete list of **FtpClientException** class members, see the FtpClientException Members topic.

Public Instance Properties

HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

FtpClientException Class | SocketTools Namespace

FtpClientException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]

```
Overrides Public ReadOnly Property Message As String
```

[C#]

```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

FtpClientException Class | SocketTools Namespace

FtpClientException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception. If a network error occurs, this value is the same as the values returned by the Windows Sockets API. For more information about socket error codes, see the Windows Socket Version 2 API error code documentation in MSDN.

See Also

FtpClientException Class | SocketTools Namespace

FtpClientException Methods

The methods of the **FtpClientException** class are listed below. For a complete list of **FtpClientException** class members, see the FtpClientException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpClientException Class | SocketTools Namespace

FtpClientException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]
```
Overrides Public Function ToString() As String
```

[C#]
```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

FtpClientException Class | SocketTools Namespace

FtpServer Class

Implements a server that enables the application to send and receive files using the File Transfer Protocol.

For a list of all members of this type, see FtpServer Members.

System.Object
  SocketTools.FtpServer

[Visual Basic]
```
Public Class FtpServer
    Implements IDisposable
```

[C#]
```
public class FtpServer : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The **SocketTools.FtpServer** class provides an interface for implementing an embedded, lightweight server that can be used to exchange files with a client using the standard File Transfer Protocol. The server can accept connections from any third-party application or a program developed using the **SocketTools.FtpClient** class.

The application specifies an initial server configuration by setting the relevant properties and can implement event handlers to monitor the activities of the clients that have connected to the server. The class automatically handles the standard FTP commands and requires minimal coding on the part of the application that is hosting the control. However, the application may also use event mechanism to filter specific commands or to extend the protocol by providing custom implementations of existing commands or add entirely new commands.

The server supports active and passive mode file transfers, has compatibility options for NAT router and firewall support, and provides support for secure file transfers using explicit TLS sessions. Secure connections require that a valid server certificate be installed on the system.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

FtpServer Members | SocketTools Namespace

FtpServer Members

FtpServer overview

Public Static (Shared) Methods

ErrorText	Returns the description of an error code.

Public Instance Constructors

FtpServer Constructor	Initializes a new instance of the FtpServer class.

Public Instance Fields

AdapterAddress	Returns the IP address associated with the specified network adapter.

Public Instance Properties

AdapterCount	Get the number of available local and remote network adapters.
AuthFail	Gets and sets the maximum number of authentication attempts permitted.
AuthTime	Gets and sets the amount of time a client has to authenticate the session.
CertificateName	Gets and sets a value that specifies the name of the server certificate.
CertificatePassword	Gets and sets the password associated with the server certificate.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateUser	Gets and sets the user that owns the server certificate.
ClientAccess	Gets and sets the access rights that have been granted to the client session.
ClientAddress	Return the Internet address of the current client connection.
ClientCount	Return the number of active client sessions connected to the server.
ClientDirectory	Return the current working directory for the active client session.
ClientHome	Return the home directory for the active client session.
ClientHost	Return the host name that the client used to establish the connection.
ClientId	Gets the unique client identifier for the current client session.

ClientIdle	Gets and sets the maximum number of seconds a client can be idle before the server terminates the session.
ClientPort	Gets a value that specifies the port number used by the current client session.
ClientThread	Gets the thread ID for the current client session.
ClientUser	Return the user name associated with the specified client session.
CommandLine	Return the complete command line issued by the client.
Directory	Get and set the full path to the root directory assigned to the server.
ExecTime	Get and set maximum number of seconds that the server will permit an external command to execute.
ExternalAddress	Get and set the external IP address used for passive mode data transfers.
HiddenFiles	Determine if the server should permit access to hidden files.
Identity	Gets and sets a string that identifies the server to the client.
IdleTime	Gets and sets the maximum number of seconds a client can be idle before the server terminates the session.
IsActive	Gets a value which indicates if the server is active.
IsAnonymous	Determine if the active client session has authenticated as an anonymous user.
IsAuthenticated	Determine if the active client session has been authenticated.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsListening	Gets a value which indicates if the server is listening for client connections.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalPath	Return the full path to the local file or directory that is the target of the current command.
LocalTime	Determines if the server should return file and directory times adjusted for the local timezone.
LocalUser	Determines if the server should perform user authentication using the Windows local account

	database.
LockFiles	Determines if files should be exclusively locked when a client attempts to upload or download a file.
LogFile	Gets and sets the name of the server log file.
LogFormat	Gets and sets the format used when updating the server log file.
LogLevel	Gets and sets the level of detail included in the server log file.
MaxClients	Gets and sets the maximum number of clients that can connect to the server.
MaxGuests	Gets and sets the maximum number of anonymous users that are permitted to connect to the server.
MaxPort	Gets and sets the maximum port number used by the server for passive data connections.
MemoryUsage	Gets the amount of unmanaged memory currently allocated by the server.
MinPort	Gets and sets the maximum port number used by the server for passive data connections.
MultiUser	Determine if the server should be started in multi-user mode.
Options	Gets and sets the options that may be specified for the server instance.
Priority	Gets and sets a value which specifies the server priority.
ReadOnly	Determine if the server should prevent clients from uploading files.
Restricted	Determine if the server should be started in restricted mode, limiting client access to the server.
Secure	Determine if the server should accept secure client connections.
ServerAddress	Gets and sets the address that will be used by the server to listen for connections.
ServerHandle	Gets the handle to the server created to listen for client connections.
ServerName	Gets a value which specifies the host name for the local system.
ServerPort	Gets and sets the port number that will be used by the server to listen for connections.
ServerThread	Gets the thread ID for the current server.
ServerUuid	Gets and sets the Universally Unique Identifier

	(UUID) associated with the server.
StackSize	Gets and sets the size of the stack allocated for threads created by the server.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the network function tracing logfile.
TraceFlags	Gets and sets a value which specifies the network function tracing flags.
UnixMode	Determine if the server should impersonate a UNIX-based operating system.
Version	Gets a value which returns the current version of the FtpServer class library.
VirtualPath	Return the virtual path to the local file or directory that is the target of the current command.

Public Instance Methods

AddUser	Overloaded. Add a new virtual user to the server.
Authenticate	Overloaded. Authenticate the client and assign access rights for the session.
DeleteUser	Remove a virtual user from the server.
Disconnect	Overloaded. Disconnect the specified client session from the server.
Dispose	Overloaded. Releases all resources used by FtpServer.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the FtpServer class.
RegisterProgram	Overloaded. Register a program for use with the SITE EXEC command.
Reset	Reset the internal state of the object, resetting all properties to their default values.
ResolvePath	Overloaded. Resolve a path to its full virtual or local file name.

Restart	Restarts the server and terminates all active client connections.
Resume	Resume accepting new client connections.
SendResponse	Overloaded. Send a result code and message to the client in response to a command.
Start	Overloaded. Start listening for client connections on the specified IP address and port number.
Stop	Stop listening for new client connections and terminate all active clients already connected to the server.
Suspend	Suspend accepting new client connections.
Throttle	Overloaded. Limit the maximum number of client connections, connections per IP address and connection rate.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the server.

Public Instance Events

OnAuthenticate	Occurs when the client has requested authentication with the specified username and password.
OnCommand	Occurs when a client has issued a command to the server.
OnConnect	Occurs when a connection is established with the remote host.
OnDisconnect	Occurs when the remote host disconnects from the local system.
OnDownload	Occurs when a connection is established with the remote host.
OnError	Occurs when an network operation fails.
OnExecute	Occurs when the client has executed an external program on the server.
OnIdle	Occurs when the there are no clients connected to the server.
OnLogin	Occurs when the client has successfully authenticated the session.
OnLogout	Occurs when the client has logged out or reinitialized the session.
OnResult	Occurs when the command issued by the client has been processed by the server.
OnStart	Occurs when the server starts accepting connections.

OnStop	Occurs when the server stops accepting connections.
OnTimeout	Occurs when the client has exceeded the maximum allowed idle time.
OnUpload	Occurs when the client has successfully uploaded a file to the server.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the FtpServer class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpServer Class | SocketTools Namespace

FtpServer Constructor

Initializes a new instance of the FtpServer class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FtpServer();
```

See Also

FtpServer Class | SocketTools Namespace

FtpServer Fields

The fields of the **FtpServer** class are listed below. For a complete list of **FtpServer** class members, see the FtpServer Members topic.

Public Instance Fields

◆ AdapterAddress	Returns the IP address associated with the specified network adapter.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.AdapterAddress Field

Returns the IP address associated with the specified network adapter.

[Visual Basic]
```
Public ReadOnly AdapterAddress As AdapterAddressArray
```

[C#]
```
public readonly AdapterAddressArray AdapterAddress;
```

Remarks

The **AdapterAddress** array returns the IP addresses that are associated with the local network or remote dial-up network adapters configured on the system. The **AdapterCount** property can be used to determine the number of adapters that are available.

Multihomed systems with more than one local network adapter, or a combination of local and dial-up adapters will not be listed in a specific order. An application should not make the assumption that the first address returned by **AdapterAddress** always refers to a local network adapter.

Note that it is possible that the **AdapterCount** property will return 0, and **AdapterAddress** will return an empty string. This indicates that the system does not have a physical network adapter with an assigned IP address, and there are no dial-up networking connections currently active. If a dial-up networking connection is established at some later point, the **AdapterCount** property will change to 1, and the **AdapterAddress** property will return the IP address allocated for that connection.

See Also

FtpServer Class | SocketTools Namespace | AdapterAddressArray Class | AdapterCount Property

FtpServer Properties

The properties of the **FtpServer** class are listed below. For a complete list of **FtpServer** class members, see the FtpServer Members topic.

Public Instance Properties

AdapterCount	Get the number of available local and remote network adapters.
AuthFail	Gets and sets the maximum number of authentication attempts permitted.
AuthTime	Gets and sets the amount of time a client has to authenticate the session.
CertificateName	Gets and sets a value that specifies the name of the server certificate.
CertificatePassword	Gets and sets the password associated with the server certificate.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateUser	Gets and sets the user that owns the server certificate.
ClientAccess	Gets and sets the access rights that have been granted to the client session.
ClientAddress	Return the Internet address of the current client connection.
ClientCount	Return the number of active client sessions connected to the server.
ClientDirectory	Return the current working directory for the active client session.
ClientHome	Return the home directory for the active client session.
ClientHost	Return the host name that the client used to establish the connection.
ClientId	Gets the unique client identifier for the current client session.
ClientIdle	Gets and sets the maximum number of seconds a client can be idle before the server terminates the session.
ClientPort	Gets a value that specifies the port number used by the current client session.
ClientThread	Gets the thread ID for the current client session.
ClientUser	Return the user name associated with the specified client session.

CommandLine	Return the complete command line issued by the client.
Directory	Get and set the full path to the root directory assigned to the server.
ExecTime	Get and set maximum number of seconds that the server will permit an external command to execute.
ExternalAddress	Get and set the external IP address used for passive mode data transfers.
HiddenFiles	Determine if the server should permit access to hidden files.
Identity	Gets and sets a string that identifies the server to the client.
IdleTime	Gets and sets the maximum number of seconds a client can be idle before the server terminates the session.
IsActive	Gets a value which indicates if the server is active.
IsAnonymous	Determine if the active client session has authenticated as an anonymous user.
IsAuthenticated	Determine if the active client session has been authenticated.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsListening	Gets a value which indicates if the server is listening for client connections.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalPath	Return the full path to the local file or directory that is the target of the current command.
LocalTime	Determines if the server should return file and directory times adjusted for the local timezone.
LocalUser	Determines if the server should perform user authentication using the Windows local account database.
LockFiles	Determines if files should be exclusively locked when a client attempts to upload or download a file.
LogFile	Gets and sets the name of the server log file.
LogFormat	Gets and sets the format used when updating the server log file.
LogLevel	Gets and sets the level of detail included in the

	server log file.
MaxClients	Gets and sets the maximum number of clients that can connect to the server.
MaxGuests	Gets and sets the maximum number of anonymous users that are permitted to connect to the server.
MaxPort	Gets and sets the maximum port number used by the server for passive data connections.
MemoryUsage	Gets the amount of unmanaged memory currently allocated by the server.
MinPort	Gets and sets the maximum port number used by the server for passive data connections.
MultiUser	Determine if the server should be started in multi-user mode.
Options	Gets and sets the options that may be specified for the server instance.
Priority	Gets and sets a value which specifies the server priority.
ReadOnly	Determine if the server should prevent clients from uploading files.
Restricted	Determine if the server should be started in restricted mode, limiting client access to the server.
Secure	Determine if the server should accept secure client connections.
ServerAddress	Gets and sets the address that will be used by the server to listen for connections.
ServerHandle	Gets the handle to the server created to listen for client connections.
ServerName	Gets a value which specifies the host name for the local system.
ServerPort	Gets and sets the port number that will be used by the server to listen for connections.
ServerThread	Gets the thread ID for the current server.
ServerUuid	Gets and sets the Universally Unique Identifier (UUID) associated with the server.
StackSize	Gets and sets the size of the stack allocated for threads created by the server.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Trace	Gets and sets a value which indicates if network function logging is enabled.

TraceFile	Gets and sets a value which specifies the name of the network function tracing logfile.
TraceFlags	Gets and sets a value which specifies the network function tracing flags.
UnixMode	Determine if the server should impersonate a UNIX-based operating system.
Version	Gets a value which returns the current version of the FtpServer class library.
VirtualPath	Return the virtual path to the local file or directory that is the target of the current command.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.AdapterCount Property

Get the number of available local and remote network adapters.

[Visual Basic]

```
Public ReadOnly Property AdapterCount As Integer
```

[C#]

```
public int AdapterCount {get;}
```

Property Value

Returns the number of available local and remote network adapters.

Remarks

The **AdapterCount** property returns the number of local and remote dial-up networking adapters available on the local system. This value can be used in conjunction with the **AdapterAddress** array to enumerate the IP addresses assigned to the various network adapters.

Note that it is possible that the **AdapterCount** property will return 0, and **AdapterAddress** will return an empty string. This indicates that the system does not have a physical network adapter with an assigned IP address, and there are no dial-up networking connections currently active. If a dial-up networking connection is established at some later point, the **AdapterCount** property will change to 1, and the **AdapterAddress** property will return the IP address allocated for that connection.

See Also

FtpServer Class | SocketTools Namespace | AdapterAddress Field

FtpServer.AuthFail Property

Gets and sets the maximum number of authentication attempts permitted.

[Visual Basic]

```
Public Property AuthFail As Integer
```

[C#]

```
public int AuthFail {get; set;}
```

Property Value

An integer value that specifies the maximum number of user authentication attempts permitted.

Remarks

The **AuthFail** property value specifies the maximum number of user authentication attempts that are permitted until the server terminates the client connection. A value of zero specifies that the default configuration limit of 3 authentication attempts per login should be allowed. The maximum number of authentication attempts is 10.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.AuthTime Property

Gets and sets the amount of time a client has to authenticate the session.

[Visual Basic]

```
Public Property AuthTime As Integer
```

[C#]

```
public int AuthTime {get; set;}
```

Property Value

An integer value that specifies the user authentication time in seconds.

Remarks

The **AuthTime** property value specifies the maximum number of user authentication attempts that are permitted until the server terminates the client connection. A value of zero specifies the default value of 60 seconds. If the value is non-zero, the minimum value is 20 seconds and the maximum value is 300 seconds (5 minutes). This value is used to ensure that a client has successfully authenticated itself within a limited period of time. This prevents a potential denial-of-service attack against the server where clients establish connections and hold them open without authentication. In conjunction with the **AuthFail** property, this also limits the ability of a client to attempt to probe the server for valid username and password combinations.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.CertificateName Property

Gets and sets a value that specifies the name of the server certificate.

[Visual Basic]

```
Public Property CertificateName As String
```

[C#]

```
public string CertificateName {get; set;}
```

Property Value

A string which specifies the server certificate name.

Remarks

The **CertificateName** property sets the common name or friendly name of the certificate that should be used when starting a secure server. If the **Secure** property is set to True, this property must be specify a valid certificate name. The certificate must have a private key associated with it, otherwise client connections will fail because the class will be unable to create a security context for the session.

When the certificate store is searched for a matching certificate, it will first search for any certificate with a friendly name that matches the property value. If no valid certificate is found, it will then search for a certificate with a matching common name.

Certificates may be installed and viewed on the local system using the Certificate Manager that is included with the Windows operating system. For more information, refer to the documentation for the Microsoft Management Console.

See Also

FtpServer Class | SocketTools Namespace | CertificateStore Property | Secure Property

FtpServer.CertificateStore Property

Gets and sets a value that specifies the name of the local certificate store.

[Visual Basic]

```
Public Property CertificateStore As String
```

[C#]

```
public string CertificateStore {get; set;}
```

Property Value

A string which specifies the certificate store name. The default value is the current user's personal certificate store.

Remarks

The **CertificateStore** property is used to specify the name of the certificate store which contains the security certificate to use when security is enabled for the server. The certificate may either be stored in the registry or in a file. If the certificate is stored in the registry, then this property should be set to one of the following predefined values:

Store Name	Description
CA	Certification authority certificates. These are certificates that are issued by entities which are entrusted to issue certificates to other individuals or organizations. Companies such as VeriSign and Thawte act as certification authorities.
MY	Personal certificates and their associated private keys for the current user. This store typically holds the client certificates used to establish a user's credentials. If a certificate store is not specified, this is the default value that is used.
ROOT	Certificates that have been self-signed by a certificate authority. Root certificates for a number of different certification authorities such as VeriSign and Thawte are installed as part of the operating system and periodically updated by Microsoft.

In most cases the server certificate will be installed in the user's personal certificate store, and therefore it is not necessary to set this property value because that is the default location that will be used to search for the certificate. This property is only used if the **CertificateName** property is also set to a valid certificate name.

If you are using a local certificate store, with the certificate and private key stored in the registry, you can explicitly specify whether the certificate store for the current user or the local machine (all users) should be used. This is done by prefixing the certificate store name with "HKCU" for the current user, or "HKLM" for the local machine. For example, a certificate store name of "HKLM:MY" would specify the personal certificate store for the local machine, rather than the current user. If neither prefix is specified, it will default to the certificate store for the current user.

This property may also be used to specify a file that contains the server certificate. In this case, the property should specify the full path to the file and must contain both the certificate and private key in

PKCS12 format. If the file is protected by a password, the **CertificatePassword** property must also be set to specify the password.

See Also

FtpServer Class | SocketTools Namespace | CertificatePassword Property | Secure Property

FtpServer.ClientAccess Property

Gets and sets the access rights that have been granted to the client session.

[Visual Basic]
```
Public Property ClientAccess As UserAccess
```

[C#]
```
public FtpServer.UserAccess ClientAccess {get; set;}
```

Property Value

A UserAccess enumeration that specifies on or more user access permissions.

Remarks

The **ClientAccess** property is used to determine all of the access permissions that are currently granted to an authenticated client session and optionally change those permissions. For a list of user access rights that can be granted to the client, see the UserAccess enumeration.

When modifying the value of this property, it is recommended that you use bitwise OR and AND operands to set and clear specific bitflags. The exception is when using the **ftpAccessDefault** permission. If you wish to reset the client session to use the default permissions based on the server configuration and client authentication, then you should assign this value directly to the **ClientAccess** property.

This property should only be accessed within an event handler such as **OnCommand** because its value is specific to the client session that raised the event. This property will always return a value of zero outside of an event handler, and an exception will be raised if you attempt to modify this property outside of an event handler.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.ClientAddress Property

Return the Internet address of the current client connection.

[Visual Basic]

```
Public ReadOnly Property ClientAddress As String
```

[C#]

```
public string ClientAddress {get;}
```

Property Value

A string that specifies the client Internet Protocol address.

Remarks

The **ClientAddress** property returns the address of the current client session which has connected to the server. This property should only be accessed within an event handler such as **OnConnect** because its value is specific to the client session that raised the event. This property will always return an empty string when accessed outside of an event handler.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.ClientCount Property

Return the number of active client sessions connected to the server.

[Visual Basic]
```
Public ReadOnly Property ClientCount As Integer
```

[C#]
```
public int ClientCount {get;}
```

Property Value

An integer value that specifies the number of active client sessions.

Remarks

The **ClientCount** read-only property returns the number of active client sessions that have been established with the server. The value includes both authenticated and unauthenticated client sessions.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.ClientDirectory Property

Return the current working directory for the active client session.

[Visual Basic]

```
Public ReadOnly Property ClientDirectory As String
```

[C#]

```
public string ClientDirectory {get;}
```

Property Value

A string that specifies the full path to a local directory on the server.

Remarks

The **ClientDirectory** property returns the current working directory for the active client session. Initially this value will be the absolute path on the local system that maps to an authenticated client's home directory. The client can change its current working directory using the CWD command. The **ClientHome** property will return the home directory that has been assigned to the client.

It is important to note that the current working directory for client sessions is virtual, and does not reflect the current working directory for the server process. This property should only be accessed within an event handler after the client session has been authenticated. Unauthenticated clients are not assigned a current working directory. This property will always return an empty string when accessed outside of an event handler.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.ClientHome Property

Return the home directory for the active client session.

[Visual Basic]

```
Public ReadOnly Property ClientHome As String
```

[C#]

```
public string ClientHome {get;}
```

Property Value

A string that specifies the full path to a local directory on the server.

Remarks

The **ClientHome** property returns the home directory for the active client session. This will be the same path to the home directory specified when the **Authenticate** method was used to authenticate the client session. If a home directory was not explicitly assigned when the client was authenticated, then this property returns the default home directory that was created for the client, or the server root directory if the **MultiUser** property was set to **False** when the server was started. The **ClientDirectory** property will return the current working directory for the client.

This property should only be accessed within an event handler after the client session has been authenticated. Unauthenticated clients are not assigned a home directory. This property will always return an empty string when accessed outside of an event handler.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.ClientHost Property

Return the host name that the client used to establish the connection.

[Visual Basic]

```
Public ReadOnly Property ClientHost As String
```

[C#]

```
public string ClientHost {get;}
```

Property Value

A string that specifies the host name used by the client to connect to the server.

Remarks

The **ClientHost** property returns the host name that the client used to establish the connection. If the client sends the HOST command, this property will return the value specified by the client. If the client does not explicitly specify the host name, then this property will return the same host name that was assigned to the server when it started.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.ClientId Property

Gets the unique client identifier for the current client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which uniquely identifies the client session.

Remarks

Each client connection that is accepted by the server is assigned a unique numeric value. This value can be used by the application to identify that client session, and is different than the socket handle allocated for the client. While it is possible for a client socket handle to be reused by the operating system, client IDs are unique throughout the life of the server session and are never duplicated.

It is important to note that the actual value of the client ID should be considered opaque. It is only guaranteed that the value will be greater than zero, and that it will be unique to the client session.

This property only returns a meaningful value when accessed from within a class event handler, or a method that has been invoked from within an event handler.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.ClientIdle Property

Gets and sets the maximum number of seconds a client can be idle before the server terminates the session.

[Visual Basic]
```
Public Property ClientIdle As Integer
```

[C#]
```
public int ClientIdle {get; set;}
```

Property Value

An integer value that specifies the idle timeout period in seconds.

Remarks

The **ClientIdle** property returns the maximum number of seconds that the active client session may be idle before the server closes the control connection. The idle timeout period for each client session is based on the value of the **IdleTime** property when the server was started, with the default value of 300 seconds (5 minutes). Changing this value inside an event handler will change the timeout period for the active client session. Clients may also use the SITE IDLE command to request that the server change the idle timeout period.

This property should only be accessed within an event handler such as **OnConnect** or **OnLogin** because its value is specific to the client session that raised the event. This property will always return a value of zero outside of an event handler, and an exception will be raised if you attempt to modify this property outside of an event handler.

When the timeout period for the client has elapsed, the **OnTimeout** event will fire prior to the client being disconnected from the server.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.ClientPort Property

Gets a value that specifies the port number used by the current client session.

[Visual Basic]

```
Public ReadOnly Property ClientPort As Integer
```

[C#]

```
public int ClientPort {get;}
```

Property Value

An integer value which specifies the peer port number.

Remarks

The **ClientPort** property returns the port number that the current client has used when establishing a connection with the server. This property value is only meaningful when accessed within an event handler such as the **OnConnect** event.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.ClientThread Property

Gets the thread ID for the current client session.

[Visual Basic]
```
Public ReadOnly Property ClientThread As Integer
```

[C#]
```
public int ClientThread {get;}
```

Property Value

An integer value which identifies the client thread that was created to manage the client session.

Remarks

Until the thread terminates, the thread identifier uniquely identifies the thread throughout the system.

This property only returns a meaningful value when accessed from within a class event handler, or a method that has been invoked from within an event handler.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.ClientUser Property

Return the user name associated with the specified client session.

[Visual Basic]

```
Public ReadOnly Property ClientUser As String
```

[C#]

```
public string ClientUser {get;}
```

Property Value

A string that specifies the user name associated with the active client session.

Remarks

The **ClientUser** property returns the user name that the client used to authenticate the client session. This property should only be accessed within an event handler after the client session has been authenticated. Unauthenticated clients are not assigned a user name. This property will always return an empty string when accessed outside of an event handler.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.Directory Property

Get and set the full path to the root directory assigned to the server.

[Visual Basic]

```
Public Property Directory As String
```

[C#]

```
public string Directory {get; set;}
```

Property Value

A string that specifies the full path to a local directory on the server.

Remarks

The **Directory** property returns the path to the root directory for the server. If this property is set to the name of a valid directory before the server is started, that directory will be considered the root directory for the server. If this property is not set, or is set to an empty string, then the server will use the current working directory as its root directory, however this is not recommended. It is recommended that you specify an absolute path to the directory, otherwise the path will be relative to the current working directory. You may include environment variables in the path surrounded by percent (%) symbols and they will be expanded.

If you have configured the server to permit clients to upload files, you must ensure that your application has permission to create files in the directory that you specify. A recommended location for the server root directory would be a subdirectory of the %ALLUSERSPROFILE% directory. Using the environment variable ensures that your server will work correctly on different versions of Windows. If the root directory does not exist at the time that the server is started, it will be created.

If the **MultiUser** property is **False**, all authenticated clients will have their current working directory initialized to the server root directory. If the **MultiUser** property is **True**, then the Public and User subdirectories will be created in the root directory, and each authenticated client will have their current working directory initialized to their individual home directory.

This property can be read after the server has started and it will return the full path to the root directory. However, attempting to change the value of this property after the server has started will cause an exception to be raised. To change the root directory for the server, you must first call the **Stop** method which will terminate all active client connections.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.ExternalAddress Property

Get and set the external IP address used for passive mode data transfers.

[Visual Basic]
```
Public Property ExternalAddress As String
```

[C#]
```
public string ExternalAddress {get; set;}
```

Property Value

A string that specifies an Internet Protocol address.

Remarks

When using passive mode file transfers, the server creates a second listening (passive) socket that is used to exchange data between the client and server. The client sends the PASV command, and the server responds with its IP address and the ephemeral port number that was selected for the transfer. If the server is located behind a router that performs Network Address Translation (NAT), the address that the server will return will typically be a non-routable IP address assigned to the local system on the LAN side of the network. Setting the **ExternalAddress** property will instruct the server to return a different IP address in response to the PASV command sent by the client. Typically you would use the address assigned to the router on the Internet side of the connection.

If the **ExternalAddresss** property is not assigned a specific address, reading this property value will cause the control to automatically determine its external IP address. This requires that you have an active connection to the Internet; checking the value of this property on a system that uses dial-up networking may cause the operating system to automatically connect to the Internet service provider. The control may be unable to determine the external IP address for the local host for a number of reasons, particularly if the system is behind a firewall or uses a proxy server that restricts access to external sites on the Internet. If the external address for the local host cannot be determined, the property will return an empty string.

This property will not change the IP address the server is using to listen for client connections. The only way to change the listening IP address is to stop and restart the server using the new address. This property only changes the IP address that is reported to clients when a passive data connection is used. Incorrect use of this property can prevent the client from establishing a data connection to the server. The address must be in the same address family as the local address that the server was started with. For example, if the server was started using an IPv4 address, the IP address assigned to this property cannot be an IPv6 address.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.HiddenFiles Property

Determine if the server should permit access to hidden files.

[Visual Basic]

```
Public Property HiddenFiles As Boolean
```

[C#]

```
public bool HiddenFiles {get; set;}
```

Property Value

A Boolean value that specifies if hidden files can be accessed by clients.

Remarks

The **HiddenFiles** property determines if the server should allow clients to access files with the hidden and/or system attribute. If this property is **True**, then hidden files are included in directory listings and clients may download or replace hidden files. If the property is **False**, hidden files are not included in directory listings and any attempt to access, delete or modify a hidden file will result in an error.

The default value for this property is **False.**

See Also

FtpServer Class | SocketTools Namespace

FtpServer.Identity Property

Gets and sets a string that identifies the server to the client.

[Visual Basic]
```
Public Property Identity As String
```

[C#]
```
public string Identity {get; set;}
```

Property Value

A string that identifies the server instance.

Remarks

The **Identity** property returns a string that is used to identify the server. It is used for informational purposes only and does not affect the operation of the server. Typically the string specifies the name of the application and a version number, and is displayed whenever a client establishes its initial connection to the server. This property can be set to assign an identity to the server, however after the server has started this property becomes read-only.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.IdleTime Property

Gets and sets the maximum number of seconds a client can be idle before the server terminates the session.

[Visual Basic]

```
Public Property IdleTime As Integer
```

[C#]

```
public int IdleTime {get; set;}
```

Property Value

An integer value that specifies the idle timeout period in seconds.

Remarks

The **IdleTime** property specifies the maximum number of seconds that a client session may be idle before the server closes the control connection to the client. A value of zero specifies the default value of 300 seconds (5 minutes). If the value is non-zero, the minimum value is 60 seconds and the maximum value is 7200 seconds (2 hours). This value is used to initialize the default idle timeout period for each client session. A client may request that the server change the idle timeout period for its session by sending the SITE IDLE command. The server determines if a client is idle based on the time the last command was issued and whether or not a file transfer is in progress.

The **ClientIdle** property can be used to determine the idle timeout period for a specific client. When the timeout period for the client has elapsed, the **OnTimeout** event will fire prior to the client being disconnected from the server.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.IsActive Property

Gets a value which indicates if the server is active.

[Visual Basic]

```
Public ReadOnly Property IsActive As Boolean
```

[C#]

```
public bool IsActive {get;}
```

Property Value

A Boolean value that specifies if the server instance is currently active.

Remarks

The **IsActive** property returns **True** if the server has been started using the **Start** method. If the server has not been started, the property will return **False**.

To determine if the server is accepting client connections, use the **IsListening** property. This property will only indicate if the server has been started. For example, if the server has been suspended using the **Suspend** method, this property will return a value of **True**, while the **IsListening** property will return a value of **False**.

An application should not depend on this property returning **False** immediately after the **Stop** method has been called to shutdown the server. This property will continue to return **True** until all clients have disconnected from the server and the server thread has terminated. To determine when the server has stopped, implement a handler for the **OnStop** event.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.IsAnonymous Property

Determine if the active client session has authenticated as an anonymous user.

[Visual Basic]

```
Public ReadOnly Property IsAnonymous As Boolean
```

[C#]

```
public bool IsAnonymous {get;}
```

Property Value

A Boolean value that specifies if the active client session is anonymous.

Remarks

The **IsAnonymous** property returns **True** if the active client session has authenticated as an anonymous (guest) user. This property should only be accessed within an event handler such as **OnCommand** because its value is specific to the client session that raised the event. This property will always return a value of **False** outside of an event handler.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.IsAuthenticated Property

Determine if the active client session has been authenticated.

[Visual Basic]

```
Public ReadOnly Property IsAuthenticated As Boolean
```

[C#]

```
public bool IsAuthenticated {get;}
```

Property Value

A Boolean value that specifies if the active client session has been authenticated.

Remarks

The **IsAuthenticated** property returns **True** if the active client session has successfully authenticated with a valid username and password. This property should only be accessed within an event handler such as **OnCommand** because its value is specific to the client session that raised the event. This property will always return a value of **False** outside of an event handler.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.IsInitialized Property

Gets a value which indicates if the current instance of the class has been initialized successfully.

[Visual Basic]
```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]
```
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the class instance has been initialized; otherwise returns **false**.

Remarks

The **IsInitialized** property is used to determine if the current instance of the class has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the class may not be able to initialize itself.

The most common reasons that a class instance may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required networking libraries or not being able to access the system registry.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.IsListening Property

Gets a value which indicates if the server is listening for client connections.

[Visual Basic]

```
Public ReadOnly Property IsListening As Boolean
```

[C#]

```
public bool IsListening {get;}
```

Property Value

Returns **true** if the server is listening for client connections; otherwise returns **false**.

Remarks

The **IsListening** property will return **true** if the **Start** method was called and the server is currently accepting incoming client connections. In all other situations, this property will return **false**.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public FtpServer.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.LocalPath Property

Return the full path to the local file or directory that is the target of the current command.

[Visual Basic]

```
Public Property LocalPath As String
```

[C#]

```
public string LocalPath {get; set;}
```

Property Value

A string that specifies the full path to a local file or directory on the server.

Remarks

The **LocalPath** property returns the full path to a local file name or directory specified by the client as an argument to a standard FTP command. For example, if the client sends the RETR command to the server, this property will return the complete path to the local file that the client wants to download. This property will only return a value for those standard commands that perform some action on a file or directory, otherwise it will return an empty string.

Setting this property allows you to effectively redirect the client to use a different file than the one that was actually requested. If the path is absolute, then it will be used as-is. If the path is relative, it will be relative to the current working directory for the active client session. The full path to this file is not limited to the server root directory or its subdirectory, it can specify a file anywhere on the local system. If this property is set to an empty string, then the server will revert to using the actual file or directory name specified by the command.

This property should only be set within an **OnCommand** event handler, and only for those commands that perform an action on a file or directory. If the current command does not target a file or directory, setting this property will cause an exception to be raised by the control. Exercise caution when using this property to redirect the server to use a different file than the one requested by the client; changing the target file may cause the client to behave in unexpected ways.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.LocalTime Property

Determines if the server should return file and directory times adjusted for the local timezone.

[Visual Basic]

```
Public Property LocalTime As Boolean
```

[C#]

```
public bool LocalTime {get; set;}
```

Property Value

A Boolean value that specifies of files should be listed using the local timezone on the server.

Remarks

The **LocalTime** property determines if the server should return file and directory times adjusted for the local timezone. By default, the server will return all file times as UTC values. This option affects the time information sent to a client when a list of files is requested, as well as when status information is requested for a specific file. This property value will not affect the MDTM and MFMT commands which always use file times as UTC values.

The default value for this property is **False**.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.LocalUser Property

Determines if the server should perform user authentication using the Windows local account database.

[Visual Basic]

```
Public Property LocalUser As Boolean
```

[C#]

```
public bool LocalUser {get; set;}
```

Property Value

A Boolean value that specifies if the server should authenticate local users.

Remarks

The **LocalUser** property determines if the server should perform user authentication using the Windows local account database. If this option is not specified, the application is responsible for creating virtual users using the **AddUser** method or implementing an **OnAuthenticate** event handler and authenticating client sessions individually.

If this property is set to **True**, a client can authenticate as a local user, however the session will not inherit that user's access rights. All files that are accessed or created by the server will continue to use the permissions of the process that started the server. For example, consider a server application that was started by local user ***A***. Next, a client connects to the server and authenticates itself as local user ***B***. When that client uploads a file to the server, the file that is created will be owned by user ***A***, not user ***B***. This ensures that the server application retains ownership and control of the files that have been created or modified.

The default value for this property is **False**.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.LockFiles Property

Determines if files should be exclusively locked when a client attempts to upload or download a file.

[Visual Basic]

```
Public Property LockFiles As Boolean
```

[C#]

```
public bool LockFiles {get; set;}
```

Property Value

A Boolean value that specifies if files should be locked during file transfers.

Remarks

The **LocalTime** property determines if files should be exclusively locked when a client attempts to upload or download a file. If another client attempts to access the same file, the operation will fail. By default, the server will permit multiple clients to access the same file, although it will still write-lock files that are in the process of being uploaded..

The default value for this property is **False**.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.LogFile Property

Gets and sets the name of the server log file.

[Visual Basic]

```
Public Property LogFile As String
```

[C#]

```
public string LogFile {get; set;}
```

Property Value

A string that specifies the full path to a local log file.

Remarks

The **LogFile** property is used to specify the name of a file that will contain a log of all client activity. The **LogFormat** and **LogLevel** properties affect the specific format for the file and the level of detail included in the log. It is recommended that you specify an absolute path to the log file, otherwise the path will be relative to the current working directory. You may include environment variables in the path surrounded by percent (%) symbols and they will be expanded.

If the log file does not exist it will be created when the server is started. If file already exists, the server will append the new logging data to the file. The server must have permission to create and/or modify the specified file.

Setting this property to an empty string after the server has been started will have the effect of disabling logging, setting the logging level to 0 and the logging format to **FormatType.formatNone**.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.LogFormat Property

Gets and sets the format used when updating the server log file.

[Visual Basic]

```
Public Property LogFormat As FormatType
```

[C#]

```
public FtpServer.FormatType LogFormat {get; set;}
```

Property Value

A FormatType enumeration that specifies the log file format.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.LogLevel Property

Gets and sets the level of detail included in the server log file.

[Visual Basic]

```
Public Property LogLevel As Integer
```

[C#]

```
public int LogLevel {get; set;}
```

Property Value

An integer value that specifies the amount of information the server writes to the log file.

Remarks

The **LogLevel** property is used to specify the level of detail that should generated in the log file. The minimum value is 1 and the maximum value is 10. If this parameter is zero, it is the same as specifying a log file format of **ftpLogNone** and will disable logging by the server.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.MaxClients Property

Gets and sets the maximum number of clients that can connect to the server.

[Visual Basic]

```
Public Property MaxClients As Integer
```

[C#]

```
public int MaxClients {get; set;}
```

Property Value

An integer value which specifies the maximum number of client sessions that will be accepted by the server. A value of zero specifies that there is no fixed limit to the maximum number of clients.

Remarks

The **MaxClients** property specifies the maximum number of client connections that will be accepted by the server. Once the maximum number of connections has been established, the server will reject any subsequent connections until the number of active client connections drops below the specified value. A value of zero specifies that there should be no limit on the number of clients.

Changing the value of this property while a server is actively listening for connections will modify the maximum number of client connections permitted, but it will not affect connections that have already been established.

By default, there are no limits on the number of client connections or the connection rate when a server is started. Use the **Throttle** method to change the maximum number of client connections per IP address or the overall connection rate threshold for the server.

It is important to note that regardless of the maximum number of clients specified by this property, the actual number of client connections that can be managed by the server depends on the number of sockets that can be allocated from the operating system. The amount of physical memory installed on the system affects the number of connections that can be maintained because each connection allocates memory for the socket context from the non-paged memory pool.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.MaxGuests Property

Gets and sets the maximum number of anonymous users that are permitted to connect to the server.

[Visual Basic]

```
Public Property MaxGuests As Integer
```

[C#]

```
public int MaxGuests {get; set;}
```

Property Value

An integer value that specifies the maximum number of anonymous (guest) users.

Remarks

The **MaxGuests** property specifies the maximum number of guest users that will be accepted by the server. Once the maximum number of connections has been established, the server will reject any subsequent connections until the number of active guest users drops below the specified value. A guest user is one that authenticates with the username "anonymous" and their email address as the password.

Changing the value of this property while a server is actively listening for connections will modify the maximum number of guest logins permitted, but it will not affect connections that have already been established. You can also use the **Throttle** method to change the maximum number of clients, the maximum number of clients per IP address and the rate at which clients can connect to the server.

The default value for this property is zero, disabling guest logins. If your server is accessible to the public and you decide to allow guest users, it is recommended that you set the **Restricted** property to True, and you should not grant permission for guests to upload files or execute registered programs using the SITE EXEC command.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.MaxPort Property

Gets and sets the maximum port number used by the server for passive data connections.

[Visual Basic]

```
Public Property MaxPort As Integer
```

[C#]

```
public int MaxPort {get; set;}
```

Property Value

An integer value that specifies the maximum port number.

Remarks

The **MaxPort** property specifies the maximum range of port numbers that will be used with passive data connections. A value of zero specifies the default value of 65535 should be used. The minimum value of this member is 10000 and the maximum value is 65535. If the value is non-zero, it must be greater than the value of the **MinPort** property.

Attempting to change the value of this property after the server has started will cause an exception to be raised. To change the maximum port number for the server, you must first call the **Stop** method which will terminate all active client connections.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.MinPort Property

Gets and sets the maximum port number used by the server for passive data connections.

[Visual Basic]

```
Public Property MinPort As Integer
```

[C#]

```
public int MinPort {get; set;}
```

Property Value

An integer value that specifies the minimum port number.

Remarks

The **MinPort** property specifies the minimum range of port numbers that will be used with passive data connections. A value of zero specifies that the default value of 30000 should be used. The minimum value of this member is 10000 and the maximum value is 65535. If the value is non-zero, it must be less than the value of the **MaxPort** property.

Attempting to change the value of this property after the server has started will cause an exception to be raised. To change the minimum port number for the server, you must first call the **Stop** method which will terminate all active client connections.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.MemoryUsage Property

Gets the amount of unmanaged memory currently allocated by the server.

[Visual Basic]

```
Public ReadOnly Property MemoryUsage As Long
```

[C#]

```
public long MemoryUsage {get;}
```

Property Value

A long integer which specifies the number of bytes of memory allocated.

Remarks

This read-only property returns the amount of memory allocated by the server and all active client sessions. It enumerates all unmanaged memory allocations made by the server process and client session threads, returning the total number of bytes allocated for the server. This value reflects the amount of memory explicitly allocated by the class and does not reflect the total working set size of the process, or the memory allocated on the managed heap which is used by the .NET garbage collector.

Getting the value of this property forces the server into a locked state, and all client sessions will block while the memory usage is being calculated. Because this enumerates all unmanaged heaps allocated for the server process, it can be an expensive operation, particularly when there are a large number of active clients connected to the server. Frequently checking the value of this property can significantly degrade the performance of the server. It is primarily intended for use as a debugging tool to determine if memory usage is the result of an increase in active client sessions. If the value returned by this property remains reasonably constant, but the amount of memory allocated for the process continues to grow, it could indicate a memory leak in some other area of the application.

See Also

FtpServer Class | SocketTools Namespace | StackSize Property

FtpServer.MultiUser Property

Determine if the server should be started in multi-user mode.

[Visual Basic]

```
Public Property MultiUser As Boolean
```

[C#]

```
public bool MultiUser {get; set;}
```

Property Value

A Boolean value that specifies if the server should be started in multi-user mode.

Remarks

The **MultiUser** property determines if the server should be started in multi-user mode. If this property is set to **True**, each user will be assigned their own home directory which will be based on their user name. When a client authenticates as that user, its current working directory is set to the user's home directory. If this property is set to **False**, then all users will share the server root directory by default. This property does not affect the maximum number of simultaneous client connections to the server. To isolate users to their own individual home directory, set the **Restricted** property to **True**.

Setting this property to **True** will cause the server to create two subdirectories under the server root directory named Public and Users. The Public subdirectory is where public files should be stored, and also serves as the home directory for anonymous (guest) users. The Users subdirectory is where the home directories for each user will be created.

Attempting to change the value of this property after the server has started will cause an exception to be raised. To change this property value, you must first call the **Stop** method which will terminate all active client connections.

The default value for this property is **False**.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.Options Property

Gets and sets the options that may be specified for the server instance.

[Visual Basic]

```
Public Property Options As ServerOptions
```

[C#]

```
public FtpServer.ServerOptions Options {get; set;}
```

Property Value

A ServerOptions enumeration that specifies one or more server options.

Remarks

The **Options** property is used to specify one or more server options as bitflags using the ServerOptions enumeration. Each option has a corresponding property, and it is recommended that you use those properties, such as **LocalUser** and **UnixMode**, to specify whether a particular server option should be enabled or disabled. Using the class properties will make your code more readable and ensure forward compatibility.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.Priority Property

Gets and sets a value which specifies the server priority.

[Visual Basic]
```
Public Property Priority As ServerPriority
```

[C#]
```
public FtpServer.ServerPriority Priority {get; set;}
```

Property Value

Returns a ServerPriority enumeration value which specifies the current server priority. The default value for this property is **priorityNormal**.

Remarks

The **Priority** property can be used to control the processor usage, memory and network bandwidth allocated by the server for client sessions. The default priority balances resource utilization and client throughput while ensuring that the user interface remains responsive to the user. Lower priorities reduce the overall resource utilization at the expense of throughput.

Higher priority values increases the thread priority and processor utilization for the client sessions. It is not recommended that you increase the server priority unless you understand the implications of doing so and have thoroughly tested your application. Raising the priority of the server can have a negative impact on the responsiveness of the user interface.

See Also

FtpServer Class | SocketTools Namespace | ServerPriority Enumeration

FtpServer.ReadOnly Property

Determine if the server should prevent clients from uploading files.

[Visual Basic]

```
Public Property ReadOnly As Boolean
```

[C#]

```
public bool ReadOnly {get; set;}
```

Property Value

A Boolean value that specifies if clients have read-only access to the server.

Remarks

The **ReadOnly** property determines if the server should only allow read-only access to files by default, changing the default permissions granted to authenticated users. If this property is set to **True**, anonymous users will not be able to upload, rename or delete files and cannot create subdirectories. This is recommended if the server is publicly accessible over the Internet and guest logins are permitted.

Attempting to change the value of this property after the server has started will cause an exception to be raised. To change this property value, you must first call the **Stop** method which will terminate all active client connections.

The default value for this property is **False**.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.Restricted Property

Determine if the server should be started in restricted mode, limiting client access to the server.

[Visual Basic]

```
Public Property Restricted As Boolean
```

[C#]

```
public bool Restricted {get; set;}
```

Property Value

A Boolean value that specifies if client access to the server is restricted.

Remarks

The **Restricted** property determines if the server should be initialized in a restricted mode that isolates the server and limits the ability for clients to access files on the host system. If this property is set to **True**, all file transfers are limited to the user's home directory and certain site-specific commands are disabled. This is recommended for general purpose applications designed to accept connections from clients over the Internet. This property value is only meaningful if the **MultiUser** property has also been set to **True**.

Attempting to change the value of this property after the server has started will cause an exception to be raised. To change this property value, you must first call the **Stop** method which will terminate all active client connections.

The default value for this property is **False**.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.Secure Property

Determine if the server should accept secure client connections.

[Visual Basic]
```
Public Property Secure As Boolean
```

[C#]
```
public bool Secure {get; set;}
```

Property Value

A Boolean value that specifies if the server should accept secure connections.

Remarks

The **Secure** property determines if client connections are encrypted using the standard SSL or TLS security protocols. The default value for this property is **False**, which specifies that clients will use a standard, unencrypted connection to the server. To enable secure connections, the application should set this property value to **True** prior to calling the **Start** method.

When secure connections are enabled, the server will accept the client connection and then wait for the client to initiate the handshake where both the client and server negotiate the various encryption options available. This process is handled automatically by the server, and all that is required is that the application specify the server certificate which should be used. This is done by setting the **CertificateName** property, and optionally the **CertificateStore** property if required.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.ServerAddress Property

Gets and sets the address that will be used by the server to listen for connections.

[Visual Basic]

```
Public Property ServerAddress As String
```

[C#]

```
public string ServerAddress {get; set;}
```

Property Value

A string which specifies the IP address that the server will use to listen for incoming client connections. An empty string indicates that the server will accept connections on any valid network interface configured for the local system.

Remarks

The **ServerAddress** property is used to specify the default address that the server will use when listening for connections. Setting this property to the value 0.0.0.0 or an empty string indicates that the server should listen for client connections using any valid network interface. If an address is specified, it must be a valid Internet address that is bound to a network adapter configured on the local system. Clients will only be able to connect to the server using that specific address.

It is common to set this property to the value 127.0.0.1 for testing purposes. It is a non-routable address that specifies the local system, and most software firewalls are configured so they do not block applications using this address.

See Also

FtpServer Class | SocketTools Namespace | ServerName Property | ServerPort Property

FtpServer.ServerName Property

Gets a value which specifies the host name for the local system.

[Visual Basic]

```
Public Property ServerName As String
```

[C#]

```
public string ServerName {get; set;}
```

Property Value

A string which specifies the local host name.

Remarks

The **ServerName** property returns the fully-qualified host name assigned to the local system. This consists of the local computer name and its domain name. The actual value returned depends on the system configuration. If no domain has been specified for the system, then only the machine name will be returned.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.ServerPort Property

Gets and sets the port number that will be used by the server to listen for connections.

[Visual Basic]

```
Public Property ServerPort As Integer
```

[C#]

```
public int ServerPort {get; set;}
```

Property Value

An integer value which specifies the port number.

Remarks

The **ServerPort** property is used to set the port number that server will use to listen for incoming client connections. Valid port numbers are in the range of 1 to 65535. It is recommended that most custom servers specify a port number larger than 5000 to avoid potential conflicts with standard Internet services and ephemeral ports used by client applications.

If a port number is specified that is already in use by another application, the **OnError** event will fire and the background server thread will terminate. To enable a server to be stopped and immediately restarted using the same address and port number, make sure that the **ReuseAddress** property is set to a value of **true**.

See Also

FtpServer Class | SocketTools Namespace | ServerAddress Property | ServerName Property

FtpServer.ServerThread Property

Gets the thread ID for the current server.

[Visual Basic]

```
Public ReadOnly Property ServerThread As Integer
```

[C#]

```
public int ServerThread {get;}
```

Property Value

An integer value which identifies the server thread that was created. A return value of zero specifies that no server has been started.

Remarks

Until the thread terminates, the thread identifier uniquely identifies the thread throughout the system.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.ServerUuid Property

Gets and sets the Universally Unique Identifier (UUID) associated with the server.

[Visual Basic]

```
Public Property ServerUuid As String
```

[C#]

```
public string ServerUuid {get; set;}
```

Property Value

A string that specifies the UUID assigned to the server instance.

Remarks

The **ServerUuid** property returns the UUID that uniquely identifies this instance of the server. If the application does not set this property, a temporary UUID will be assigned to the server. If a value is assigned to this property, it must be a valid UUID string. A permanent UUID can be generated using a utility such as **uuidgen** which is included with Visual Studio.

Attempting to change the value of this property after the server has started will cause an exception to be raised. To change this property value, you must first call the **Stop** method which will terminate all active client connections.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.StackSize Property

Gets and sets the size of the stack allocated for threads created by the server.

[Visual Basic]

```
Public Property StackSize As Integer
```

[C#]

```
public int StackSize {get; set;}
```

Property Value

An integer value that specifies the initial amount of memory that is committed to the stack for each thread created by the server.

Remarks

The default stack size for each thread is set to 256K for 32-bit processes and 512K for 64-bit processes. Increasing or decreasing the stack size will only affect new threads that are created by the server, it will not affect those threads that have already been created to manage active client sessions. It is recommended that most applications use the default stack size.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]

```
Public Property ThrowError As Boolean
```

[C#]

```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.Trace Property

Gets and sets a value which indicates if network function logging is enabled.

[Visual Basic]

```
Public Property Trace As Boolean
```

[C#]

```
public bool Trace {get; set;}
```

Property Value

Returns **true** if network function tracing is enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Trace** property is used to enable (or disable) the tracing of network function calls. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the SocketTools networking classes and is enabled or disabled for an entire process. This means that once trace logging is enabled for a given component, all of the function calls made by the process using any of the SocketTools classes will be logged. For example, if you have an application using both the File Transfer Protocol and Post Office Protocol classes, and you set the **Trace** property to **true**, function calls made by both classes will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all classes used by the process.

If trace logging is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the logfile is fairly large. Since logfiles can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

When redistributing your application, make sure that you include the **SocketTools10.TraceLog.dll** module with your installation. If this library is not present, then no trace output will be generated and the value of the **Trace** property will be ignored. Only those function calls made by the SocketTools networking classes will be logged. Calls made directly to the Windows Sockets API, or calls made by other classes, will not be logged.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.TraceFile Property

Gets and sets a value which specifies the name of the network function tracing logfile.

[Visual Basic]

```
Public Property TraceFile As String
```

[C#]

```
public string TraceFile {get; set;}
```

Property Value

A string which specifies the name of the file.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named **SocketTools.log** is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFile** property should be set to the same value for each instance. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
 MyApp INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0 MyApp WRN:
connect(46, 192.0.0.1:1234, 16) returned -1 [10035] MyApp ERR: accept(46,
NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the control is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Note that if the specified file cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.TraceFlags Property

Gets and sets a value which specifies the network function tracing flags.

[Visual Basic]

```
Public Property TraceFlags As TraceOptions
```

[C#]

```
public FtpServer.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration which specifies the amount of detail written to the trace logfile.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Because network function tracing is enabled per-process, the trace flags are shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFlags** property should be set to the same value for each instance. Changing the trace flags for any one instance of the class will affect the logging performed for all SocketTools classes used by the application.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being written and the error **errorOperationWouldBlock** occurs, a warning is generated because the application simply needs to attempt to write the data at a later time.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.UnixMode Property

Determine if the server should impersonate a UNIX-based operating system.

[Visual Basic]

```
Public Property UnixMode As Boolean
```

[C#]

```
public bool UnixMode {get; set;}
```

Property Value

A Boolean value that specifies if the server should impersonate a UNIX based server.

Remarks

The **UnixMode** property determines if the server should impersonate a UNIX-based operating system. If this property is set to a value of **True**, the server will identify itself as running on a UNIX system and directory listings will be in a format commonly used by UNIX. If this property value is **False**, the server will identify itself as running on Windows NT and directory listings will be in the same format used by the Microsoft IIS FTP server. Note that this option does not affect the path delimiter used with file and directory names.

Attempting to change the value of this property after the server has started will cause an exception to be raised. To change this property value, you must first call the **Stop** method which will terminate all active client connections.

The default value for this property is **False**.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.Version Property

Gets a value which returns the current version of the FtpServer class library.

[Visual Basic]

```
Public ReadOnly Property Version As String
```

[C#]

```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the FtpServer class library. This value can be used by an application for validation and debugging purposes.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.VirtualPath Property

Return the virtual path to the local file or directory that is the target of the current command.

[Visual Basic]

```
Public Property VirtualPath As String
```

[C#]

```
public string VirtualPath {get; set;}
```

Property Value

A string that specifies the virtual path to the local file accessed by the active client session.

Remarks

The **VirtualPath** property returns the virtual path to a local file name or directory specified by the client as an argument to a standard FTP command. For example, if the client sends the RETR command to the server, this property will return the complete virtual path to the file that the client wants to download. This property will only return a value for those standard commands that perform some action on a file or directory, otherwise it will return an empty string.

Setting this property allows you to effectively redirect the client to use a different file than the one that was actually requested. If the path is absolute, then it will be used as-is. If the path is relative, it will be relative to the current working directory for the active client session. If this property is set to an empty string, then the server will revert to using the actual file or directory name specified by the command.

This property should only be set within an **OnCommand** event handler, and only for those commands that perform an action on a file or directory. If the current command does not target a file or directory, setting this property will cause an exception to be raised by the control. Exercise caution when using this property to redirect the server to use a different file than the one requested by the client; changing the target file may cause the client to behave in unexpected ways.

See Also

FtpServer Class | SocketTools Namespace

FtpServer Methods

The methods of the **FtpServer** class are listed below. For a complete list of **FtpServer** class members, see the FtpServer Members topic.

Public Static (Shared) Methods

ErrorText	Returns the description of an error code.

Public Instance Methods

AddUser	Overloaded. Add a new virtual user to the server.
Authenticate	Overloaded. Authenticate the client and assign access rights for the session.
DeleteUser	Remove a virtual user from the server.
Disconnect	Overloaded. Disconnect the specified client session from the server.
Dispose	Overloaded. Releases all resources used by FtpServer.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the FtpServer class.
RegisterProgram	Overloaded. Register a program for use with the SITE EXEC command.
Reset	Reset the internal state of the object, resetting all properties to their default values.
ResolvePath	Overloaded. Resolve a path to its full virtual or local file name.
Restart	Restarts the server and terminates all active client connections.
Resume	Resume accepting new client connections.
SendResponse	Overloaded. Send a result code and message to the client in response to a command.
Start	Overloaded. Start listening for client connections on the specified IP address and port number.
Stop	Stop listening for new client connections and terminate all active clients already connected to the server.
Suspend	Suspend accepting new client connections.

Throttle	Overloaded. Limit the maximum number of client connections, connections per IP address and connection rate.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the server.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the FtpServer class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.AddUser Method

Add a new virtual user to the server.

Overload List

Add a new virtual user to the server.

public bool AddUser(string,string);

Add a new virtual user to the server.

public bool AddUser(string,string,UserAccess,string);

Add a new virtual user to the server.

public bool AddUser(string,string,string);

See Also

FtpServer Class | SocketTools Namespace

FtpServer.AddUser Method (String, String, UserAccess, String)

Add a new virtual user to the server.

[Visual Basic]

```
Overloads Public Function AddUser( _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal accessFlags As UserAccess, _
   ByVal homeDirectory As String _
) As Boolean
```

[C#]

```
public bool AddUser(
   string userName,
   string userPassword,
   UserAccess accessFlags,
   string homeDirectory
);
```

Parameters

userName

A string which specifies the user name. The maximum length of a username is 63 characters and it is recommended that names be limited to alphanumeric characters. Whitespace, control characters and certain symbols such as path delimiters and wildcard characters are not permitted. If an invalid character is included in the name, the method will fail with an error indicating the username is invalid. The username must be at least three characters in length. Usernames are not case sensitive.

userPassword

A string which specifies the user password. The maximum length of a password is 63 characters and is limited to printable characters. Whitespace and control characters are not permitted. If an invalid character is included in the password, the function will fail with an error indicating the password is invalid. The password must be at least one character in length. Passwords are case sensitive.

accessFlags

A UserAccess enumeration which specifies the access clients will be given when authenticated as this user.

homeDirectory

A string which specifies the directory that will be the virtual user's home directory. If the server was started in multi-user mode, this directory will be relative to the user directory created by the server, otherwise it will be relative to the server root directory. If the directory does not exist, it will be created the first time that the virtual user successfully logs in to the server. If this parameter is an empty string, a default home directory will be created for the virtual user.

Return Value

A boolean value which specifies if the virtual user was added to the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **AddUser** method creates a virtual user that is associated with the server. When a client connects with the server and provides authentication credentials, the server will check if the username has been created using this method. If a match is found, the client access rights will be updated.

If you wish to modify the information for an existing user, it is not necessary to delete the username first. If this method is called with a username that already exists, that record is replaced with the values passed to this method. You cannot use this method to create a virtual user named "anonymous".

The virtual users created by this method exist only as long as the server is active. If you wish to maintain a persistent database of users and passwords, you are responsible for its implementation based on the requirements of your specific application. For example, a simple implementation would be to store the user information in a local XML or INI file and then read that configuration file after the server has started, calling this method for each user that is listed.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.AddUser Overload List

FtpServer.AddUser Method (String, String, String)

Add a new virtual user to the server.

[Visual Basic]

```
Overloads Public Function AddUser( _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal homeDirectory As String _
) As Boolean
```

[C#]

```
public bool AddUser(
   string userName,
   string userPassword,
   string homeDirectory
);
```

Parameters

userName

A string which specifies the user name. The maximum length of a username is 63 characters and it is recommended that names be limited to alphanumeric characters. Whitespace, control characters and certain symbols such as path delimiters and wildcard characters are not permitted. If an invalid character is included in the name, the method will fail with an error indicating the username is invalid. The username must be at least three characters in length. Usernames are not case sensitive.

userPassword

A string which specifies the user password. The maximum length of a password is 63 characters and is limited to printable characters. Whitespace and control characters are not permitted. If an invalid character is included in the password, the function will fail with an error indicating the password is invalid. The password must be at least one character in length. Passwords are case sensitive.

homeDirectory

A string which specifies the directory that will be the virtual user's home directory. If the server was started in multi-user mode, this directory will be relative to the user directory created by the server, otherwise it will be relative to the server root directory. If the directory does not exist, it will be created the first time that the virtual user successfully logs in to the server. If this parameter is an empty string, a default home directory will be created for the virtual user.

Return Value

A boolean value which specifies if the virtual user was added to the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **AddUser** method creates a virtual user that is associated with the server. When a client connects with the server and provides authentication credentials, the server will check if the username has been created using this method. If a match is found, the client access rights will be updated.

If you wish to modify the information for an existing user, it is not necessary to delete the username first. If this method is called with a username that already exists, that record is replaced with the values passed to this method. You cannot use this method to create a virtual user named "anonymous".

The virtual users created by this method exist only as long as the server is active. If you wish to maintain a persistent database of users and passwords, you are responsible for its implementation based on the requirements of your specific application. For example, a simple implementation would be to store the user information in a local XML or INI file and then read that configuration file after the server has started, calling this method for each user that is listed.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.AddUser Overload List

FtpServer.AddUser Method (String, String)

Add a new virtual user to the server.

[Visual Basic]

```
Overloads Public Function AddUser( _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]

```
public bool AddUser(
   string userName,
   string userPassword
);
```

Parameters

userName

A string which specifies the user name. The maximum length of a username is 63 characters and it is recommended that names be limited to alphanumeric characters. Whitespace, control characters and certain symbols such as path delimiters and wildcard characters are not permitted. If an invalid character is included in the name, the method will fail with an error indicating the username is invalid. The username must be at least three characters in length. Usernames are not case sensitive.

userPassword

A string which specifies the user password. The maximum length of a password is 63 characters and is limited to printable characters. Whitespace and control characters are not permitted. If an invalid character is included in the password, the function will fail with an error indicating the password is invalid. The password must be at least one character in length. Passwords are case sensitive.

Return Value

A boolean value which specifies if the virtual user was added to the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **AddUser** method creates a virtual user that is associated with the server. When a client connects with the server and provides authentication credentials, the server will check if the username has been created using this method. If a match is found, the client access rights will be updated.

If you wish to modify the information for an existing user, it is not necessary to delete the username first. If this method is called with a username that already exists, that record is replaced with the values passed to this method. You cannot use this method to create a virtual user named "anonymous".

The virtual users created by this method exist only as long as the server is active. If you wish to maintain a persistent database of users and passwords, you are responsible for its implementation based on the requirements of your specific application. For example, a simple implementation would be to store the user information in a local XML or INI file and then read that configuration file after the server has started, calling this method for each user that is listed.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.AddUser Overload List

FtpServer.Authenticate Method

Authenticate the client and assign access rights for the session.

Overload List

Authenticate the client and assign access rights for the session.

public bool Authenticate(UserAccess,string);

Authenticate the client and assign access rights for the session.

public bool Authenticate(int,UserAccess,string);

See Also

FtpServer Class | SocketTools Namespace

FtpServer.Authenticate Method (Int32, UserAccess, String)

Authenticate the client and assign access rights for the session.

[Visual Basic]

```
Overloads Public Function Authenticate( _
   ByVal clientId As Integer, _
   ByVal accessFlags As UserAccess, _
   ByVal homeDirectory As String _
) As Boolean
```

[C#]

```
public bool Authenticate(
   int clientId,
   UserAccess accessFlags,
   string homeDirectory
);
```

Parameters

clientId

An integer that identifies the client session.

accessFlags

A UserAccess enumeration which specifies the access clients will be given when authenticated as this user.

homeDirectory

A string which specifies the directory that will be the client's home directory. If the server was started in multi-user mode, this directory will be relative to the user directory created by the server, otherwise it will be relative to the server root directory. If the directory does not exist, it will be created. If this parameter is an empty string, a default home directory will be created for the client.

Return Value

A boolean value which specifies if the virtual user has been authenticated. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Authenticate** method authenticates a client session, typically in response to an **OnAuthenticate** event that indicates a client has requested authentication. It is recommended that most applications specify **accessDefault** as the *accessFlags* parameter for a client session, since this allows the server automatically grant the appropriate access based on the server configuration options for normal and anonymous users. If the server is going to be publicly accessible or third-party FTP clients will be used to access the server, you should always grant the **ftpAccessList** permission to clients. Many client applications will not function correctly if they are unable to obtain a list of files in the user's home directory.

If the server was started with the **MultiUser** and **Restricted** properties set to a value of **true**, the client session will be effectively locked to its home directory and cannot navigate to the server root directory. By default, restricted client sessions are also limited to only downloading files and requesting directory listings. If a client session is not restricted, the client can access files outside of its home directory. Regardless of this option, a client cannot access files outside of the server root directory.

If the **Restricted** property is **true** or the **accessAnonymous** permission is specified, the client session will be authenticated in a restricted mode and the access rights for the session will persist until the client disconnects from the server. Unlike regular users, the access rights for a restricted client cannot be changed by the server at a later point. This restriction is designed to prevent the inadvertent granting of rights to an untrusted client that could compromise the security of the server.

If the ***homeDirectory*** parameter is an empty string and the server has been started in multi-user mode, each user is assigned their own home directory based on their username. If the server has not been started in multi-user mode, then the default home directory will be the server root directory and is shared by all users. The **ClientHome** property will return the full path to the home directory for an authenticated client.

If the **accessExecute** permission is granted to the client session, it can execute external programs using the SITE EXEC command. Because the program is executed in the context of the server process, it is recommended that you limit access to this functionality and ensure that the programs being executed do not introduce any security risks to the operating system. This permission is never granted by default, and the SITE EXEC command will return an error if the client session is anonymous, regardless of whether this permission is granted or not.

This method is should only be used for custom authentication schemes and is not necessary if you have used the **AddUser** method to create virtual users.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.Authenticate Overload List

FtpServer.Authenticate Method (UserAccess, String)

Authenticate the client and assign access rights for the session.

[Visual Basic]

```
Overloads Public Function Authenticate( _
   ByVal accessFlags As UserAccess, _
   ByVal homeDirectory As String _
) As Boolean
```

[C#]

```
public bool Authenticate(
   UserAccess accessFlags,
   string homeDirectory
);
```

Parameters

accessFlags

A UserAccess enumeration which specifies the access clients will be given when authenticated as this user.

homeDirectory

A string which specifies the directory that will be the client's home directory. If the server was started in multi-user mode, this directory will be relative to the user directory created by the server, otherwise it will be relative to the server root directory. If the directory does not exist, it will be created. If this parameter is an empty string, a default home directory will be created for the client.

Return Value

A boolean value which specifies if the virtual user has been authenticated. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Authenticate** method authenticates the active client session, typically in response to an **OnAuthenticate** event that indicates a client has requested authentication. It is recommended that most applications specify **accessDefault** as the *accessFlags* parameter for a client session, since this allows the server automatically grant the appropriate access based on the server configuration options for normal and anonymous users. If the server is going to be publicly accessible or third-party FTP clients will be used to access the server, you should always grant the **ftpAccessList** permission to clients. Many client applications will not function correctly if they are unable to obtain a list of files in the user's home directory.

If the server was started with the **MultiUser** and **Restricted** properties set to a value of **true**, the client session will be effectively locked to its home directory and cannot navigate to the server root directory. By default, restricted client sessions are also limited to only downloading files and requesting directory listings. If a client session is not restricted, the client can access files outside of its home directory. Regardless of this option, a client cannot access files outside of the server root directory.

If the **Restricted** property is **true** or the **accessAnonymous** permission is specified, the client session will be authenticated in a restricted mode and the access rights for the session will persist until the client disconnects from the server. Unlike regular users, the access rights for a restricted client cannot be changed by the server at a later point. This restriction is designed to prevent the inadvertent granting of rights to an untrusted client that could compromise the security of the server.

If the *homeDirectory* parameter is an empty string and the server has been started in multi-user mode, each user is assigned their own home directory based on their username. If the server has not been started in multi-user mode, then the default home directory will be the server root directory and is shared by all users. The **ClientHome** property will return the full path to the home directory for an authenticated client.

If the **accessExecute** permission is granted to the client session, it can execute external programs using the SITE EXEC command. Because the program is executed in the context of the server process, it is recommended that you limit access to this functionality and ensure that the programs being executed do not introduce any security risks to the operating system. This permission is never granted by default, and the SITE EXEC command will return an error if the client session is anonymous, regardless of whether this permission is granted or not.

This method is should only be used for custom authentication schemes and is not necessary if you have used the **AddUser** method to create virtual users.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.Authenticate Overload List

FtpServer.DeleteUser Method

Remove a virtual user from the server.

[Visual Basic]

```
Public Function DeleteUser( _
   ByVal userName As String _
) As Boolean
```

[C#]

```
public bool DeleteUser(
   string userName
);
```

Parameters

userName

A string which specifies the user name to be deleted. Usernames are not case sensitive.

Return Value

A boolean value which specifies if the virtual user has been deleted. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **DeleteUser** method removes a virtual user that was created by a previous call to the **AddUser** method. This method will not match partial usernames and wildcard characters cannot be used to delete multiple users. Usernames are not case sensitive. You cannot use this method to delete the "anonymous" user.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.Disconnect Method

Disconnect the specified client session from the server.

Overload List

Disconnect the specified client session from the server.

public void Disconnect();

Disconnect the specified client session from the server.

public void Disconnect(int);

See Also

FtpServer Class | SocketTools Namespace

FtpServer.Disconnect Method (Int32)

Disconnect the specified client session from the server.

[Visual Basic]

```
Overloads Public Sub Disconnect( _
   ByVal clientId As Integer _
)
```

[C#]

```
public void Disconnect(
   int clientId
);
```

Parameters

clientId

An integer that identifies the client session.

Return Value

A boolean value which specifies if the client has been signaled to disconnect from the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Disconnect** method terminates the specified client session, releasing the socket handle other resources that were allocated for the session. It is only necessary to use this method if you want the server to explicitly terminate a client connection. Normally the client will close its connection to the server, the **OnDisconnect** event will fire and the server will automatically disconnect the client.

This method signals the thread that is managing the client that it should disconnect from the server, and it will begin the process of terminating the session. This is an asynchronous process and it is not guaranteed that the client will have actually disconnected from the server at the time that this method returns to the caller.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.Disconnect Overload List

FtpServer.Disconnect Method ()

Disconnect the specified client session from the server.

[Visual Basic]
```
Overloads Public Sub Disconnect()
```

[C#]
```
public void Disconnect();
```

Return Value

A boolean value which specifies if the client has been signaled to disconnect from the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Disconnect** method terminates the active client session, releasing the socket handle other resources that were allocated for the session. It is only necessary to use this method if you want the server to explicitly terminate a client connection. Normally the client will close its connection to the server, the **OnDisconnect** event will fire and the server will automatically disconnect the client.

This method signals the thread that is managing the client that it should disconnect from the server, and it will begin the process of terminating the session. This is an asynchronous process and it is not guaranteed that the client will have actually disconnected from the server at the time that this method returns to the caller.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.Disconnect Overload List

FtpServer.Dispose Method

Releases all resources used by FtpServer.

Overload List

Releases all resources used by FtpServer.

public void Dispose();

Releases the unmanaged resources allocated by the FtpServer class and optionally releases the managed resources.

protected void Dispose(bool);

See Also

FtpServer Class | SocketTools Namespace

FtpServer.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the FtpServer class and optionally releases the managed resources.

[Visual Basic]

```
Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]

```
protected void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **FtpServer** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.Dispose Overload List

FtpServer.Dispose Method ()

Releases all resources used by FtpServer.

[Visual Basic]

```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]

```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method stops the server, terminates all active client sessions and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.Dispose Overload List

FtpServer.Finalize Method

Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.

[Visual Basic]

```
Overrides Protected Sub Finalize()
```

[C#]

```
protected override void Finalize();
```

See Also

FtpServer Class | SocketTools Namespace

FtpServer.Initialize Method

Initialize an instance of the FtpServer class.

Overload List

Initialize an instance of the FtpServer class.

public bool Initialize();

Initialize an instance of the FtpServer class.

public bool Initialize(string);

See Also

FtpServer Class | SocketTools Namespace | Uninitialize Method

FtpServer.Initialize Method (String)

Initialize an instance of the FtpServer class.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the FtpServer class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of FtpServer can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the FtpServer class.

Example

The following example shows how to use the Initialize method to initialize an instance of the class. This example assumes that the license key string has been defined in code.

```
SocketTools.FtpServer server = new SocketTools.FtpServer();

if (server.Initialize(strLicenseKey) == false)
{
    MessageBox.Show(server.LastErrorString, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    return;
}
```

```
Dim Server As New SocketTools.FtpServer

If Server.Initialize(strLicenseKey) = False Then
    MsgBox(Server.LastErrorString, vbIconExclamation)
    Exit Sub
End If
```

See Also

FtpServer Class | SocketTools Namespace | FtpServer.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

FtpServer.Initialize Method ()

Initialize an instance of the FtpServer class.

[Visual Basic]

```
Overloads Public Function Initialize() As Boolean
```

[C#]

```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the FtpServer class, loading the networking library and allocating resources for the current thread. Typically it is not necessary to explicitly call this method because the instance of the class is initialized by the class constructor. However, if the **Uninitialize** method is called, the class must be re-initialized before any other methods are called.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.Initialize Overload List | Uninitialize Method

FtpServer.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. If a server has been started, it will be stopped and any active client connections will be terminated. All properties will be reset to their default values.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.Restart Method

Restarts the server and terminates all active client connections.

[Visual Basic]

```
Public Function Restart() As Boolean
```

[C#]

```
public bool Restart();
```

Return Value

A boolean value which specifies if the server was restarted. A return value of **true** specifies that the server has been successfully restarted. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Restart** method terminates all active client connections, recreates a new listening socket bound to the same address and port number, and then resumes accepting new client connections.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.RegisterProgram Method

Register a program for use with the SITE EXEC command.

Overload List

Register a program for use with the SITE EXEC command.

public bool RegisterProgram(string,string);

Register a program for use with the SITE EXEC command.

public bool RegisterProgram(string,string,string);

Register a program for use with the SITE EXEC command.

public bool RegisterProgram(string,string,string,string);

See Also

FtpServer Class | SocketTools Namespace

FtpServer.RegisterProgram Method (String, String, String, String)

Register a program for use with the SITE EXEC command.

[Visual Basic]

```
Overloads Public Function RegisterProgram( _
   ByVal command As String, _
   ByVal program As String, _
   ByVal parameters As String, _
   ByVal directory As String _
) As Boolean
```

[C#]

```
public bool RegisterProgram(
   string command,
   string program,
   string parameters,
   string directory
);
```

Parameters

command

A string which identifies the external program. This is the name that is passed to the SITE EXEC command and does not need to match the actual name of the executable file on the local system. The maximum length of the command name is 31 characters and this parameter cannot be an empty string.

program

A string which specifies the full path to the executable program on the local system.

parameters

A string that specifies additional parameters for the program. This value will be passed to the program as command line arguments. If the program does not require any command line parameters, this value may be an empty string.

directory

A string that specifies the current working directory for the program. If this value is an empty string, the server will use the root document directory assigned to the server.

Return Value

A boolean value which specifies if the external program has been registered with the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **RegisterProgram** method registers an executable program for use with the SITE EXEC command. Because this can present a significant security risk to the server, clients are not given permission to use this command by default. A client must be explicitly granted permission to use SITE EXEC by including **accessExecute** as one of the permissions when authenticating the client session with the **Authenticate** method or creating a virtual user using the **AddUser** method.

To give the server complete control over what programs can be executed using SITE EXEC, the program must be registered with the server and referenced by an alias specified by the *CommandName* parameter. The maximum length of a program name is 31 characters and it must be at least 3 characters in length. The name must only consist of alphanumeric characters and the first character of the program name cannot be numeric. The program name is not case-sensitive, however convention is to use uppercase characters. If a program name is specified that already has been registered, it will be updated with the new information provided by this method.

The ***ProgramFile*** string specifies file name of the program that will be executed. You should not install any executable programs in the server root directory or its subdirectories. A client should never have the ability to directly access the executable file itself. It is permitted to have multiple command names that reference the same executable file. The only requirement is that the command names be unique. The program name may contain environment variables surrounded by % symbols. For example, %ProgramFiles% would be expanded to the C:\Program Files folder.

It is important to note that the program specified by ***ProgramFile*** must be an executable file, not a script or batch file. If the program name does not contain a directory path, then the standard Windows pathing rules will be used when searching for an executable file that matches the given name. It is recommended that you always provide a full path to the executable file.

The ***Parameters*** string is used to define optional command line parameters that will be included with the command. This string can contain placeholders that are replaced by additional parameters specified by the client when it sends the SITE EXEC command. First replacement parameter is %1, the second is %2 and so on.

The executable program that is registered using this method must be a console application that writes to standard output. Programs that write directly to a console, or programs written to use a Windows user interface are not supported and will yield unpredictable results. In most cases, those programs that do not use standard input and output will be forcibly terminated by the server. If the program attempts to read from standard input, it will immediately encounter an end-of-file condition. Programs executed by the SITE EXEC command have no input; it is similar to a program that has its input redirected from the NUL: device. If the program must process a file on the server, the local file name should be passed as a command line parameter.

The output from the program will be redirected back to the client control channel. The output should be textual, with each line of text terminated by a carriage return and linefeed (CRLF). Programs that write binary data to standard output, particular data with embedded nulls, will yield unpredictable results and are not supported. To ensure that the program output conforms to the protocol standard, any non-printable characters will be replaced with a space and each line of output will be prefixed by a single space.

If the server is running on a system with User Account Control (UAC) enabled and does not have elevated privileges, do not register a program that requires elevated privileges or has a manifest that specifies the requestedExecutionLevel as requiring administrative privileges.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.RegisterProgram Overload List

FtpServer.RegisterProgram Method (String, String, String)

Register a program for use with the SITE EXEC command.

[Visual Basic]

```
Overloads Public Function RegisterProgram( _
   ByVal command As String, _
   ByVal program As String, _
   ByVal parameters As String _
) As Boolean
```

[C#]

```
public bool RegisterProgram(
   string command,
   string program,
   string parameters
);
```

Parameters

command

A string which identifies the external program. This is the name that is passed to the SITE EXEC command and does not need to match the actual name of the executable file on the local system. The maximum length of the command name is 31 characters and this parameter cannot be an empty string.

program

A string which specifies the full path to the executable program on the local system.

parameters

A string that specifies additional parameters for the program. This value will be passed to the program as command line arguments. If the program does not require any command line parameters, this value may be an empty string.

Return Value

A boolean value which specifies if the external program has been registered with the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **RegisterProgram** method registers an executable program for use with the SITE EXEC command. Because this can present a significant security risk to the server, clients are not given permission to use this command by default. A client must be explicitly granted permission to use SITE EXEC by including **accessExecute** as one of the permissions when authenticating the client session with the **Authenticate** method or creating a virtual user using the **AddUser** method.

To give the server complete control over what programs can be executed using SITE EXEC, the program must be registered with the server and referenced by an alias specified by the *CommandName* parameter. The maximum length of a program name is 31 characters and it must be at least 3 characters in length. The name must only consist of alphanumeric characters and the first character of the program name cannot be numeric. The program name is not case-sensitive, however convention is to use upper-case characters. If a program name is specified that already has been registered, it will be updated with the new information provided by this method.

The *ProgramFile* string specifies file name of the program that will be executed. You should not install any executable programs in the server root directory or its subdirectories. A client should never have the ability to directly access the executable file itself. It is permitted to have multiple command names that reference the same executable file. The only requirement is that the command names be unique. The program name may contain environment variables surrounded by % symbols. For example, %ProgramFiles% would be expanded to the C:\Program Files folder.

It is important to note that the program specified by ***ProgramFile*** must be an executable file, not a script or batch file. If the program name does not contain a directory path, then the standard Windows pathing rules will be used when searching for an executable file that matches the given name. It is recommended that you always provide a full path to the executable file.

The ***Parameters*** string is used to define optional command line parameters that will be included with the command. This string can contain placeholders that are replaced by additional parameters specified by the client when it sends the SITE EXEC command. First replacement parameter is %1, the second is %2 and so on.

The executable program that is registered using this method must be a console application that writes to standard output. Programs that write directly to a console, or programs written to use a Windows user interface are not supported and will yield unpredictable results. In most cases, those programs that do not use standard input and output will be forcibly terminated by the server. If the program attempts to read from standard input, it will immediately encounter an end-of-file condition. Programs executed by the SITE EXEC command have no input; it is similar to a program that has its input redirected from the NUL: device. If the program must process a file on the server, the local file name should be passed as a command line parameter.

The output from the program will be redirected back to the client control channel. The output should be textual, with each line of text terminated by a carriage return and linefeed (CRLF). Programs that write binary data to standard output, particular data with embedded nulls, will yield unpredictable results and are not supported. To ensure that the program output conforms to the protocol standard, any non-printable characters will be replaced with a space and each line of output will be prefixed by a single space.

If the server is running on a system with User Account Control (UAC) enabled and does not have elevated privileges, do not register a program that requires elevated privileges or has a manifest that specifies the requestedExecutionLevel as requiring administrative privileges.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.RegisterProgram Overload List

FtpServer.RegisterProgram Method (String, String)

Register a program for use with the SITE EXEC command.

[Visual Basic]

```
Overloads Public Function RegisterProgram( _
   ByVal command As String, _
   ByVal program As String _
) As Boolean
```

[C#]

```
public bool RegisterProgram(
   string command,
   string program
);
```

Parameters

command
A string which identifies the external program. This is the name that is passed to the SITE EXEC command and does not need to match the actual name of the executable file on the local system. The maximum length of the command name is 31 characters and this parameter cannot be an empty string.

program
A string which specifies the full path to the executable program on the local system.

Return Value

A boolean value which specifies if the external program has been registered with the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **RegisterProgram** method registers an executable program for use with the SITE EXEC command. Because this can present a significant security risk to the server, clients are not given permission to use this command by default. A client must be explicitly granted permission to use SITE EXEC by including **accessExecute** as one of the permissions when authenticating the client session with the **Authenticate** method or creating a virtual user using the **AddUser** method.

To give the server complete control over what programs can be executed using SITE EXEC, the program must be registered with the server and referenced by an alias specified by the *CommandName* parameter. The maximum length of a program name is 31 characters and it must be at least 3 characters in length. The name must only consist of alphanumeric characters and the first character of the program name cannot be numeric. The program name is not case-sensitive, however convention is to use upper-case characters. If a program name is specified that already has been registered, it will be updated with the new information provided by this method.

The *ProgramFile* string specifies file name of the program that will be executed. You should not install any executable programs in the server root directory or its subdirectories. A client should never have the ability to directly access the executable file itself. It is permitted to have multiple command names that reference the same executable file. The only requirement is that the command names be unique. The program name may contain environment variables surrounded by % symbols. For example, %ProgramFiles% would be expanded to the C:\Program Files folder.

It is important to note that the program specified by *ProgramFile* must be an executable file, not a script or batch file. If the program name does not contain a directory path, then the standard Windows pathing rules will be used when searching for an executable file that matches the given name. It is recommended that you always provide a full path to the executable file.

The ***Parameters*** string is used to define optional command line parameters that will be included with the command. This string can contain placeholders that are replaced by additional parameters specified by the client when it sends the SITE EXEC command. First replacement parameter is %1, the second is %2 and so on.

The executable program that is registered using this method must be a console application that writes to standard output. Programs that write directly to a console, or programs written to use a Windows user interface are not supported and will yield unpredictable results. In most cases, those programs that do not use standard input and output will be forcibly terminated by the server. If the program attempts to read from standard input, it will immediately encounter an end-of-file condition. Programs executed by the SITE EXEC command have no input; it is similar to a program that has its input redirected from the NUL: device. If the program must process a file on the server, the local file name should be passed as a command line parameter.

The output from the program will be redirected back to the client control channel. The output should be textual, with each line of text terminated by a carriage return and linefeed (CRLF). Programs that write binary data to standard output, particular data with embedded nulls, will yield unpredictable results and are not supported. To ensure that the program output conforms to the protocol standard, any non-printable characters will be replaced with a space and each line of output will be prefixed by a single space.

If the server is running on a system with User Account Control (UAC) enabled and does not have elevated privileges, do not register a program that requires elevated privileges or has a manifest that specifies the requestedExecutionLevel as requiring administrative privileges.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.RegisterProgram Overload List

FtpServer.ResolvePath Method

Resolve a path to its full virtual or local file name.

Overload List

Resolve a path to its full virtual or local file name.

public bool ResolvePath(int,string,ref string,bool);

Resolve a path to its full virtual or local file name.

public bool ResolvePath(string,ref string,bool);

See Also

FtpServer Class | SocketTools Namespace

FtpServer.ResolvePath Method (Int32, String, String, Boolean)

Resolve a path to its full virtual or local file name.

[Visual Basic]

```
Overloads Public Function ResolvePath( _
   ByVal clientId As Integer, _
   ByVal sourcePath As String, _
   ByRef resolvedPath As String, _
   ByVal isVirtual As Boolean _
) As Boolean
```

[C#]

```
public bool ResolvePath(
   int clientId,
   string sourcePath,
   ref string resolvedPath,
   bool isVirtual
);
```

Parameters

clientId
An integer that identifies the client session.

sourcePath
A string that specifies the name of the path to resolve. This may either be a virtual path, or a path to a local file name or directory.

resolvedPath
A string that will contain the resolved path when the method returns.

isVirtual
A Boolean parameter that specifies if the source path is a virtual path or local path.

Return Value

A boolean value which specifies if the source path could be resolved. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **ResolvePath** method is used to resolve a local file name or directory to obtain its virtual path name, or obtain the full path name of a file or directory that is mapped to a virtual path. If the ***isVirtual*** parameter is **false**, the ***sourcePath*** parameter is considered to be a path to a local file or directory and the ***resolvedPath*** parameter will contain the virtual path. If the ***isVirtual*** parameter is **true**, then the s***ourcePath*** parameter is considered to be a virtual path and the ***resolvedPath*** parameter will contain the full path to the local file or directory that the virtual path is mapped to

A virtual path for the client is either relative to the server root directory, or the client home directory if the client was authenticated as a restricted user. These virtual paths are what the client will see as an absolute path on the server. For example, if the server was configured to use "C:\ProgramData\MyServer" as the root directory, and the ***SourcePath*** parameter was specified as "C:\ProgramData\MyServer\Documents\Research", this method would return the virtual path to that directory as "/Documents/Research".

If the client session was authenticated as a restricted user, then the virtual path is always relative to the client home directory instead of the server root directory. This is because restricted users are isolated to their own home directory and any subdirectories. For example, if restricted user "John" has a home directory of "C:\ProgramData\MyServer\Users\John" and the ***SourcePath*** parameter was specified as "C:\ProgramData\MyServer\Users\John\Accounting\Projections.pdf" this method would return the virtual path as "/Accounting/Projections.pdf".

If the ***sourcePath*** parameter specifies a file or directory outside of the server root directory, this method will fail and the last error code will be set to **errorInvalidFileName**. This method can only be used with authenticated clients. If the ***clientId*** parameter specifies a client session that has not been authenticated, this method will fail and the last error code will be **errorAuthenticationRequired**.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.ResolvePath Overload List

FtpServer.ResolvePath Method (String, String, Boolean)

Resolve a path to its full virtual or local file name.

[Visual Basic]

```
Overloads Public Function ResolvePath( _
   ByVal sourcePath As String, _
   ByRef resolvedPath As String, _
   ByVal isVirtual As Boolean _
) As Boolean
```

[C#]

```
public bool ResolvePath(
   string sourcePath,
   ref string resolvedPath,
   bool isVirtual
);
```

Parameters

sourcePath
A string that specifies the name of the path to resolve. This may either be a virtual path, or a path to a local file name or directory.

resolvedPath
A string that will contain the resolved path when the method returns.

isVirtual
A Boolean parameter that specifies if the source path is a virtual path or local path.

Return Value

A boolean value which specifies if the source path could be resolved. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **ResolvePath** method is used to resolve a local file name or directory to obtain its virtual path name, or obtain the full path name of a file or directory that is mapped to a virtual path. If the ***isVirtual*** parameter is **false**, the ***sourcePath*** parameter is considered to be a path to a local file or directory and the ***resolvedPath*** parameter will contain the virtual path. If the ***isVirtual*** parameter is **true**, then the s***ourcePath*** parameter is considered to be a virtual path and the ***resolvedPath*** parameter will contain the full path to the local file or directory that the virtual path is mapped to

A virtual path for the client is either relative to the server root directory, or the client home directory if the client was authenticated as a restricted user. These virtual paths are what the client will see as an absolute path on the server. For example, if the server was configured to use "C:\ProgramData\MyServer" as the root directory, and the ***SourcePath*** parameter was specified as "C:\ProgramData\MyServer\Documents\Research", this method would return the virtual path to that directory as "/Documents/Research".

If the client session was authenticated as a restricted user, then the virtual path is always relative to the client home directory instead of the server root directory. This is because restricted users are isolated to their own home directory and any subdirectories. For example, if restricted user "John" has a home directory of "C:\ProgramData\MyServer\Users\John" and the ***SourcePath*** parameter was specified as "C:\ProgramData\MyServer\Users\John\Accounting\Projections.pdf" this method would return the virtual path as "/Accounting/Projections.pdf".

If the ***sourcePath*** parameter specifies a file or directory outside of the server root directory, this method will fail and the last error code will be set to **errorInvalidFileName**. This method can only be used with authenticated clients. If the ***clientId*** parameter specifies a client session that has not been authenticated, this method will fail and the last error code will be **errorAuthenticationRequired**.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.ResolvePath Overload List

FtpServer.Resume Method

Resume accepting new client connections.

[Visual Basic]

```
Public Function Resume() As Boolean
```

[C#]

```
public bool Resume();
```

Return Value

A boolean value which specifies if the server has resumed accepting client connections. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Resume** method instructs the server to resume accepting new client connections. Any pending client connections that were requested while the server was suspended will be accepted.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.SendResponse Method

Send a result code and message to the client in response to a command.

Overload List

Send a result code and message to the client in response to a command.

public bool SendResponse(int);

Send a result code and message to the client in response to a command.

public bool SendResponse(int,int);

Send a result code and message to the client in response to a command.

public bool SendResponse(int,int,string);

Send a result code and message to the client in response to a command.

public bool SendResponse(int,string);

See Also

FtpServer Class | SocketTools Namespace

FtpServer.SendResponse Method (Int32, Int32, String)

Send a result code and message to the client in response to a command.

[Visual Basic]

```
Overloads Public Function SendResponse( _
   ByVal clientId As Integer, _
   ByVal resultCode As Integer, _
   ByVal message As String _
) As Boolean
```

[C#]

```
public bool SendResponse(
   int clientId,
   int resultCode,
   string message
);
```

Parameters

clientId
An integer that identifies the client session.

resultCode
An integer value that specifies the command result code to be returned to the client.

message
A string value that specifies a message to be sent to the client. If this parameter is an empty string, a default message associated with the result code will be used.

Return Value

A boolean value which specifies if the response was sent to the client. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **SendResponse** method is used to respond to a command issued by the client from within an **OnCommand** event handler. Command responses are normally handled by the server as a normal part of processing a command and this method is only used if the application has implemented custom commands or wishes to modify the standard responses sent by the server. The message may be a maximum of 2048 characters and may include embedded carriage-return and linefeed characters. If no message is specified, then a default message will be sent based on the result code.

Result codes must be three digits (in the range of 100 through 999) and although this method will support the use of non-standard result codes, it is recommended that the client application use the standard codes defined in RFC 959 whenever possible. The use of non-standard result codes may cause problems with FTP clients that expect specific result codes in response to a particular command.

This method should only be called once in response to a command sent by the client. If a result code has already been sent in response to a command and this method is called, it will fail and return a value of zero. This is necessary because sending multiple result codes in response to a single command may cause unpredictable behavior by the client.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.SendResponse Overload List

FtpServer.SendResponse Method (Int32, Int32)

Send a result code and message to the client in response to a command.

[Visual Basic]

```
Overloads Public Function SendResponse( _
   ByVal clientId As Integer, _
   ByVal resultCode As Integer _
) As Boolean
```

[C#]

```
public bool SendResponse(
   int clientId,
   int resultCode
);
```

Parameters

clientId

An integer that identifies the client session.

resultCode

An integer value that specifies the command result code to be returned to the client.

Return Value

A boolean value which specifies if the response was sent to the client. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **SendResponse** method is used to respond to a command issued by the client from within an **OnCommand** event handler. Command responses are normally handled by the server as a normal part of processing a command and this method is only used if the application has implemented custom commands or wishes to modify the standard responses sent by the server. The message may be a maximum of 2048 characters and may include embedded carriage-return and linefeed characters. If no message is specified, then a default message will be sent based on the result code.

Result codes must be three digits (in the range of 100 through 999) and although this method will support the use of non-standard result codes, it is recommended that the client application use the standard codes defined in RFC 959 whenever possible. The use of non-standard result codes may cause problems with FTP clients that expect specific result codes in response to a particular command.

This method should only be called once in response to a command sent by the client. If a result code has already been sent in response to a command and this method is called, it will fail and return a value of zero. This is necessary because sending multiple result codes in response to a single command may cause unpredictable behavior by the client.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.SendResponse Overload List

FtpServer.SendResponse Method (Int32, String)

Send a result code and message to the client in response to a command.

[Visual Basic]

```
Overloads Public Function SendResponse( _
   ByVal resultCode As Integer, _
   ByVal message As String _
) As Boolean
```

[C#]

```
public bool SendResponse(
   int resultCode,
   string message
);
```

Parameters

resultCode

An integer value that specifies the command result code to be returned to the client.

message

A string value that specifies a message to be sent to the client. If this parameter is an empty string, a default message associated with the result code will be used.

Return Value

A boolean value which specifies if the response was sent to the client. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **SendResponse** method is used to respond to a command issued by the client from within an **OnCommand** event handler. Command responses are normally handled by the server as a normal part of processing a command and this method is only used if the application has implemented custom commands or wishes to modify the standard responses sent by the server. The message may be a maximum of 2048 characters and may include embedded carriage-return and linefeed characters. If no message is specified, then a default message will be sent based on the result code.

Result codes must be three digits (in the range of 100 through 999) and although this method will support the use of non-standard result codes, it is recommended that the client application use the standard codes defined in RFC 959 whenever possible. The use of non-standard result codes may cause problems with FTP clients that expect specific result codes in response to a particular command.

This method should only be called once in response to a command sent by the client. If a result code has already been sent in response to a command and this method is called, it will fail and return a value of zero. This is necessary because sending multiple result codes in response to a single command may cause unpredictable behavior by the client.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.SendResponse Overload List

FtpServer.SendResponse Method (Int32)

Send a result code and message to the client in response to a command.

[Visual Basic]

```
Overloads Public Function SendResponse( _
   ByVal resultCode As Integer _
) As Boolean
```

[C#]

```
public bool SendResponse(
   int resultCode
);
```

Parameters

resultCode

An integer value that specifies the command result code to be returned to the client.

Return Value

A boolean value which specifies if the response was sent to the client. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **SendResponse** method is used to respond to a command issued by the client from within an **OnCommand** event handler. Command responses are normally handled by the server as a normal part of processing a command and this method is only used if the application has implemented custom commands or wishes to modify the standard responses sent by the server. The message may be a maximum of 2048 characters and may include embedded carriage-return and linefeed characters. If no message is specified, then a default message will be sent based on the result code.

Result codes must be three digits (in the range of 100 through 999) and although this method will support the use of non-standard result codes, it is recommended that the client application use the standard codes defined in RFC 959 whenever possible. The use of non-standard result codes may cause problems with FTP clients that expect specific result codes in response to a particular command.

This method should only be called once in response to a command sent by the client. If a result code has already been sent in response to a command and this method is called, it will fail and return a value of zero. This is necessary because sending multiple result codes in response to a single command may cause unpredictable behavior by the client.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.SendResponse Overload List

FtpServer.Start Method

Start listening for client connections on the specified IP address and port number.

Overload List

Start listening for client connections on the specified IP address and port number.

public bool Start();

Start listening for client connections on the specified IP address and port number.

public bool Start(int);

Start listening for client connections on the specified IP address and port number.

public bool Start(string,int);

Start listening for client connections on the specified IP address and port number.

public bool Start(string,int,string);

Start listening for client connections on the specified IP address and port number.

public bool Start(string,int,string,int);

Start listening for client connections on the specified IP address and port number.

public bool Start(string,int,string,int,int);

Start listening for client connections on the specified IP address and port number.

public bool Start(string,int,string,int,int,ServerOptions);

See Also

FtpServer Class | SocketTools Namespace

FtpServer.Start Method (String, Int32, String, Int32, Int32, ServerOptions)

Start listening for client connections on the specified IP address and port number.

[Visual Basic]
```
Overloads Public Function Start( _
   ByVal localAddress As String, _
   ByVal localPort As Integer, _
   ByVal rootDirectory As String, _
   ByVal maxClients As Integer, _
   ByVal idleTime As Integer, _
   ByVal options As ServerOptions _
) As Boolean
```

[C#]
```
public bool Start(
   string localAddress,
   int localPort,
   string rootDirectory,
   int maxClients,
   int idleTime,
   ServerOptions options
);
```

Parameters

localAddress

A string which specifies the local hostname or IP address address that the server should be bound to. If this parameter is an empty string, then an appropriate address will automatically be used. If a specific address is used, the server will only accept client connections on the network interface that is bound to that address.

localPort

An integer that specifies the port number the server should use to listen for client connections. If a value of zero is specified, the server will use the standard port number 21 to listen for connections, or port 990 if the server is configured to use implicit SSL. The port number used by the application must be unique and multiple instances of a server cannot use the same port number. It is recommended that a port number greater than 5000 be used for private, application-specific implementations.

rootDirectory

A string that specifies the path to the root directory for the server. If this value is an empty string, the server will use the current working directory as the root directory.

maxClients

An integer value that specifies the maximum number of clients that may connect to the server.

idleTime

An integer value that specifies the number of seconds a client can be idle before the server terminates the session.

options

A ServerOptions enumeration that specifies one or more server options.

Return Value

A boolean value which specifies if the server was started. A return value of **true** specifies that the server has been successfully started. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **Start** method begins listening for client connections on the specified local address and port number. The server is started in its own thread and manages the client sessions independently of the calling thread.

To listen for connections on any suitable IPv4 interface, specify the special dotted-quad address "0.0.0.0". You can accept connections from clients using either IPv4 or IPv6 on the same socket by specifying the special IPv6 address "::0", however this is only supported on Windows 7 SP1 and Windows Server 2008 R2 or later platforms. If no local address is specified, then the server will only listen for connections from clients using IPv4. This behavior is by design for backwards compatibility with systems that do not have an IPv6 TCP/IP stack installed.

It is recommended that you always specify an absolute path for the server root directory, either by passing the full pathname as an argument to this method or by setting the **Directory** property. If the path includes environment variables surrounded by percent (%) symbols, they will be automatically expanded.

If you have configured the server to permit clients to upload files, you must ensure that your application has permission to create files in the directory that you specify. A recommended location for the server root directory would be a subdirectory of the %ALLUSERSPROFILE% directory. Using the environment variable ensures that your server will work correctly on different versions of Windows. If the root directory does not exist at the time that the server is started, it will be created.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.Start Overload List

FtpServer.Start Method (String, Int32, String, Int32, Int32)

Start listening for client connections on the specified IP address and port number.

[Visual Basic]

```
Overloads Public Function Start( _
   ByVal localAddress As String, _
   ByVal localPort As Integer, _
   ByVal rootDirectory As String, _
   ByVal maxClients As Integer, _
   ByVal idleTime As Integer _
) As Boolean
```

[C#]

```
public bool Start(
   string localAddress,
   int localPort,
   string rootDirectory,
   int maxClients,
   int idleTime
);
```

Parameters

localAddress

A string which specifies the local hostname or IP address address that the server should be bound to. If this parameter is an empty string, then an appropriate address will automatically be used. If a specific address is used, the server will only accept client connections on the network interface that is bound to that address.

localPort

An integer that specifies the port number the server should use to listen for client connections. If a value of zero is specified, the server will use the standard port number 21 to listen for connections, or port 990 if the server is configured to use implicit SSL. The port number used by the application must be unique and multiple instances of a server cannot use the same port number. It is recommended that a port number greater than 5000 be used for private, application-specific implementations.

rootDirectory

A string that specifies the path to the root directory for the server. If this value is an empty string, the server will use the current working directory as the root directory.

maxClients

An integer value that specifies the maximum number of clients that may connect to the server.

idleTime

An integer value that specifies the number of seconds a client can be idle before the server terminates the session.

Return Value

A boolean value which specifies if the server was started. A return value of **true** specifies that the server has been successfully started. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **Start** method begins listening for client connections on the specified local address and port number. The server is started in its own thread and manages the client sessions independently of the calling thread.

To listen for connections on any suitable IPv4 interface, specify the special dotted-quad address "0.0.0.0". You can accept connections from clients using either IPv4 or IPv6 on the same socket by specifying the special IPv6 address "::0", however this is only supported on Windows 7 SP1 and Windows Server 2008 R2 or later platforms. If no local address is specified, then the server will only listen for connections from clients using IPv4. This behavior is by design for backwards compatibility with systems that do not have an IPv6 TCP/IP stack installed.

It is recommended that you always specify an absolute path for the server root directory, either by passing the full pathname as an argument to this method or by setting the **Directory** property. If the path includes environment variables surrounded by percent (%) symbols, they will be automatically expanded.

If you have configured the server to permit clients to upload files, you must ensure that your application has permission to create files in the directory that you specify. A recommended location for the server root directory would be a subdirectory of the %ALLUSERSPROFILE% directory. Using the environment variable ensures that your server will work correctly on different versions of Windows. If the root directory does not exist at the time that the server is started, it will be created.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.Start Overload List

FtpServer.Start Method (String, Int32, String, Int32)

Start listening for client connections on the specified IP address and port number.

[Visual Basic]

```
Overloads Public Function Start( _
   ByVal localAddress As String, _
   ByVal localPort As Integer, _
   ByVal rootDirectory As String, _
   ByVal maxClients As Integer _
) As Boolean
```

[C#]

```
public bool Start(
   string localAddress,
   int localPort,
   string rootDirectory,
   int maxClients
);
```

Parameters

localAddress
A string which specifies the local hostname or IP address address that the server should be bound to. If this parameter is an empty string, then an appropriate address will automatically be used. If a specific address is used, the server will only accept client connections on the network interface that is bound to that address.

localPort
An integer that specifies the port number the server should use to listen for client connections. If a value of zero is specified, the server will use the standard port number 21 to listen for connections, or port 990 if the server is configured to use implicit SSL. The port number used by the application must be unique and multiple instances of a server cannot use the same port number. It is recommended that a port number greater than 5000 be used for private, application-specific implementations.

rootDirectory
A string that specifies the path to the root directory for the server. If this value is an empty string, the server will use the current working directory as the root directory.

maxClients
An integer value that specifies the maximum number of clients that may connect to the server.

Return Value

A boolean value which specifies if the server was started. A return value of **true** specifies that the server has been successfully started. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **Start** method begins listening for client connections on the specified local address and port number. The server is started in its own thread and manages the client sessions independently of the calling thread.

To listen for connections on any suitable IPv4 interface, specify the special dotted-quad address "0.0.0.0". You can accept connections from clients using either IPv4 or IPv6 on the same socket by specifying the special IPv6 address "::0", however this is only supported on Windows 7 SP1 and Windows Server 2008 R2 or later platforms. If no local address is specified, then the server will only listen for connections from clients using IPv4. This behavior is by design for backwards compatibility with systems that do not have an

IPv6 TCP/IP stack installed.

It is recommended that you always specify an absolute path for the server root directory, either by passing the full pathname as an argument to this method or by setting the **Directory** property. If the path includes environment variables surrounded by percent (%) symbols, they will be automatically expanded.

If you have configured the server to permit clients to upload files, you must ensure that your application has permission to create files in the directory that you specify. A recommended location for the server root directory would be a subdirectory of the %ALLUSERSPROFILE% directory. Using the environment variable ensures that your server will work correctly on different versions of Windows. If the root directory does not exist at the time that the server is started, it will be created.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.Start Overload List

FtpServer.Start Method (String, Int32, String)

Start listening for client connections on the specified IP address and port number.

[Visual Basic]

```
Overloads Public Function Start( _
   ByVal localAddress As String, _
   ByVal localPort As Integer, _
   ByVal rootDirectory As String _
) As Boolean
```

[C#]

```
public bool Start(
   string localAddress,
   int localPort,
   string rootDirectory
);
```

Parameters

localAddress

A string which specifies the local hostname or IP address address that the server should be bound to. If this parameter is an empty string, then an appropriate address will automatically be used. If a specific address is used, the server will only accept client connections on the network interface that is bound to that address.

localPort

An integer that specifies the port number the server should use to listen for client connections. If a value of zero is specified, the server will use the standard port number 21 to listen for connections, or port 990 if the server is configured to use implicit SSL. The port number used by the application must be unique and multiple instances of a server cannot use the same port number. It is recommended that a port number greater than 5000 be used for private, application-specific implementations.

rootDirectory

A string that specifies the path to the root directory for the server. If this value is an empty string, the server will use the current working directory as the root directory.

Return Value

A boolean value which specifies if the server was started. A return value of **true** specifies that the server has been successfully started. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **Start** method begins listening for client connections on the specified local address and port number. The server is started in its own thread and manages the client sessions independently of the calling thread.

To listen for connections on any suitable IPv4 interface, specify the special dotted-quad address "0.0.0.0". You can accept connections from clients using either IPv4 or IPv6 on the same socket by specifying the special IPv6 address "::0", however this is only supported on Windows 7 SP1 and Windows Server 2008 R2 or later platforms. If no local address is specified, then the server will only listen for connections from clients using IPv4. This behavior is by design for backwards compatibility with systems that do not have an IPv6 TCP/IP stack installed.

It is recommended that you always specify an absolute path for the server root directory, either by passing the full pathname as an argument to this method or by setting the **Directory** property. If the path includes environment variables surrounded by percent (%) symbols, they will be automatically expanded.

If you have configured the server to permit clients to upload files, you must ensure that your application has permission to create files in the directory that you specify. A recommended location for the server root directory would be a subdirectory of the %ALLUSERSPROFILE% directory. Using the environment variable ensures that your server will work correctly on different versions of Windows. If the root directory does not exist at the time that the server is started, it will be created.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.Start Overload List

FtpServer.Start Method (String, Int32)

Start listening for client connections on the specified IP address and port number.

[Visual Basic]

```
Overloads Public Function Start( _
   ByVal localAddress As String, _
   ByVal localPort As Integer _
) As Boolean
```

[C#]

```
public bool Start(
   string localAddress,
   int localPort
);
```

Parameters

localAddress

A string which specifies the local hostname or IP address address that the server should be bound to. If this parameter is an empty string, then an appropriate address will automatically be used. If a specific address is used, the server will only accept client connections on the network interface that is bound to that address.

localPort

An integer that specifies the port number the server should use to listen for client connections. If a value of zero is specified, the server will use the standard port number 21 to listen for connections, or port 990 if the server is configured to use implicit SSL. The port number used by the application must be unique and multiple instances of a server cannot use the same port number. It is recommended that a port number greater than 5000 be used for private, application-specific implementations.

Return Value

A boolean value which specifies if the server was started. A return value of **true** specifies that the server has been successfully started. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **Start** method begins listening for client connections on the specified local address and port number. The server is started in its own thread and manages the client sessions independently of the calling thread.

To listen for connections on any suitable IPv4 interface, specify the special dotted-quad address "0.0.0.0". You can accept connections from clients using either IPv4 or IPv6 on the same socket by specifying the special IPv6 address "::0", however this is only supported on Windows 7 SP1 and Windows Server 2008 R2 or later platforms. If no local address is specified, then the server will only listen for connections from clients using IPv4. This behavior is by design for backwards compatibility with systems that do not have an IPv6 TCP/IP stack installed.

It is recommended that you always specify an absolute path for the server root directory, either by passing the full pathname as an argument to this method or by setting the **Directory** property. If the path includes environment variables surrounded by percent (%) symbols, they will be automatically expanded.

If you have configured the server to permit clients to upload files, you must ensure that your application has permission to create files in the directory that you specify. A recommended location for the server root directory would be a subdirectory of the %ALLUSERSPROFILE% directory. Using the environment variable ensures that your server will work correctly on different versions of Windows. If the root directory does not exist at the time that the server is started, it will be created.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.Start Overload List

FtpServer.Start Method (Int32)

Start listening for client connections on the specified IP address and port number.

[Visual Basic]
```
Overloads Public Function Start( _
   ByVal localPort As Integer _
) As Boolean
```

[C#]
```
public bool Start(
   int localPort
);
```

Parameters

localPort

An integer that specifies the port number the server should use to listen for client connections. If a value of zero is specified, the server will use the standard port number 21 to listen for connections, or port 990 if the server is configured to use implicit SSL. The port number used by the application must be unique and multiple instances of a server cannot use the same port number. It is recommended that a port number greater than 5000 be used for private, application-specific implementations.

Return Value

A boolean value which specifies if the server was started. A return value of **true** specifies that the server has been successfully started. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **Start** method begins listening for client connections on the specified local address and port number. The server is started in its own thread and manages the client sessions independently of the calling thread.

To listen for connections on any suitable IPv4 interface, specify the special dotted-quad address "0.0.0.0". You can accept connections from clients using either IPv4 or IPv6 on the same socket by specifying the special IPv6 address "::0", however this is only supported on Windows 7 SP1 and Windows Server 2008 R2 or later platforms. If no local address is specified, then the server will only listen for connections from clients using IPv4. This behavior is by design for backwards compatibility with systems that do not have an IPv6 TCP/IP stack installed.

It is recommended that you always specify an absolute path for the server root directory, either by passing the full pathname as an argument to this method or by setting the **Directory** property. If the path includes environment variables surrounded by percent (%) symbols, they will be automatically expanded.

If you have configured the server to permit clients to upload files, you must ensure that your application has permission to create files in the directory that you specify. A recommended location for the server root directory would be a subdirectory of the %ALLUSERSPROFILE% directory. Using the environment variable ensures that your server will work correctly on different versions of Windows. If the root directory does not exist at the time that the server is started, it will be created.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.Start Overload List

FtpServer.Start Method ()

Start listening for client connections on the specified IP address and port number.

[Visual Basic]
```
Overloads Public Function Start() As Boolean
```

[C#]
```
public bool Start();
```

Return Value

A boolean value which specifies if the server was started. A return value of **true** specifies that the server has been successfully started. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **Start** method begins listening for client connections on the specified local address and port number. The server is started in its own thread and manages the client sessions independently of the calling thread.

To listen for connections on any suitable IPv4 interface, specify the special dotted-quad address "0.0.0.0". You can accept connections from clients using either IPv4 or IPv6 on the same socket by specifying the special IPv6 address "::0", however this is only supported on Windows 7 SP1 and Windows Server 2008 R2 or later platforms. If no local address is specified, then the server will only listen for connections from clients using IPv4. This behavior is by design for backwards compatibility with systems that do not have an IPv6 TCP/IP stack installed.

It is recommended that you always specify an absolute path for the server root directory, either by passing the full pathname as an argument to this method or by setting the **Directory** property. If the path includes environment variables surrounded by percent (%) symbols, they will be automatically expanded.

If you have configured the server to permit clients to upload files, you must ensure that your application has permission to create files in the directory that you specify. A recommended location for the server root directory would be a subdirectory of the %ALLUSERSPROFILE% directory. Using the environment variable ensures that your server will work correctly on different versions of Windows. If the root directory does not exist at the time that the server is started, it will be created.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.Start Overload List

FtpServer.Stop Method

Stop listening for new client connections and terminate all active clients already connected to the server.

[Visual Basic]
```
Public Function Stop() As Boolean
```

[C#]
```
public bool Stop();
```

Return Value

A boolean value which specifies if the server was stopped. A return value of **true** specifies that the server has been successfully stopped. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Stop** method instructs the server to stop accepting client connections, disconnects all active client connections and terminates the thread that is managing the server session. If this method is called when there is one or more clients connected to the server, it will signal each client thread to terminate and then wait for the server thread to terminate.

See Also

FtpServer Class | SocketTools Namespace | Restart Method | Resume Method | Start Method | Throttle Method

FtpServer.Suspend Method

Suspend accepting new client connections.

[Visual Basic]

```
Public Function Suspend() As Boolean
```

[C#]

```
public bool Suspend();
```

Return Value

A boolean value which specifies if the server has been suspended. A return value of **true** specifies that the server has been successfully stopped. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Suspend** method instructs the server to suspend accepting new client connections. All new clients that attempt to connect to the server will be sent a 421 "service unavailable" error code and the connection will be immediately closed. To resume accepting new client connections, call the **Resume** method. This method will not affect those clients that have already established a connection with the server before the **Suspend** method was called.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.Throttle Method

Limit the maximum number of client connections.

Overload List

Limit the maximum number of client connections.

public bool Throttle(int);

Limit the maximum number of client connections and connections per IP address.

public bool Throttle(int,int);

Limit the maximum number of client connections, connections per IP address and connection rate.

public bool Throttle(int,int,int);

Limit the maximum number of client connections, connections per IP address and connection rate.

public bool Throttle(int,int,int,int);

See Also

FtpServer Class | SocketTools Namespace

FtpServer.Throttle Method (Int32, Int32, Int32, Int32)

Limit the maximum number of client connections, connections per IP address and connection rate.

[Visual Basic]

```
Overloads Public Function Throttle( _
   ByVal maxClients As Integer, _
   ByVal maxClientsPerAddress As Integer, _
   ByVal maxGuests As Integer, _
   ByVal connectionRate As Integer _
) As Boolean
```

[C#]

```
public bool Throttle(
   int maxClients,
   int maxClientsPerAddress,
   int maxGuests,
   int connectionRate
);
```

Parameters

maxClients

An integer value that specifies the maximum number of clients that may connect to the server. A value of zero specifies that there is no fixed limit to the number of client connections.

maxClientsPerAddress

An integer value that specifies the maximum number of clients that may connect to the server from the same IP address. A value of zero specifies that there is no fixed limit to the number of client connections per address. By default, there is no limit on the number of client connections per address.

maxGuests

An integer value that specifies the maximum number of anonymous (guest) users that may be logged in at any one time.

connectionRate

An integer value that specifies a restriction on the rate of client connections, limiting the number of connections that will be accepted within that period of time. A value of zero specifies that there is no restriction on the rate of client connections. The higher this value, the fewer the number of connections that will be accepted within a specific period of time. By default, there is no limit on the client connection rate.

Return Value

A boolean value which specifies if the method was successful. A return value of **true** indicates success. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

If the maximum number of client connections or maximum number of connections per address is exceeded, the server will reject subsequent connection attempts until the number of active client sessions drops below the specified threshold. Note that adjusting these values lower than the current connection limits will not affect clients that have already connected to the server. For example, if the **Start** method is called with the maximum number of clients set to 100, and then the **Throttle** method is called lowering that value to 75, no existing client connections will be affected by the change. However, the server will not accept any new connections until the number of active clients drops below 75.

If the value of the ***maxGuests*** parameter is greater than zero, then anonymous logins will be enabled and clients can authenticate with the username "anonymous" and their email address as the password. If the parameter is set to zero, then anonymous logins will be disabled. Note that this will not affect any clients that are currently logged in, it only affects those clients that connect after the **Throttle** method has been called.

Increasing the connection rate value will force the server to slow down the rate at which it will accept incoming client connection requests. For example, setting this parameter to a value of 1000 would limit the server to accepting one client connection every second, while a value of 250 would allow the server to accept four client connections per second. Note that significantly increasing the amount of time the server must wait to accept client connections can exceed the connection backlog queue, resulting in client connections being rejected.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.Throttle Overload List

FtpServer.Throttle Method (Int32, Int32, Int32)

Limit the maximum number of client connections, connections per IP address and connection rate.

[Visual Basic]

```
Overloads Public Function Throttle( _
   ByVal maxClients As Integer, _
   ByVal maxClientsPerAddress As Integer, _
   ByVal maxGuests As Integer _
) As Boolean
```

[C#]

```
public bool Throttle(
   int maxClients,
   int maxClientsPerAddress,
   int maxGuests
);
```

Parameters

maxClients

An integer value that specifies the maximum number of clients that may connect to the server. A value of zero specifies that there is no fixed limit to the number of client connections.

maxClientsPerAddress

An integer value that specifies the maximum number of clients that may connect to the server from the same IP address. A value of zero specifies that there is no fixed limit to the number of client connections per address. By default, there is no limit on the number of client connections per address.

maxGuests

An integer value that specifies the maximum number of anonymous (guest) users that may be logged in at any one time.

Return Value

A boolean value which specifies if the method was successful. A return value of **true** indicates success. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

If the maximum number of client connections or maximum number of connections per address is exceeded, the server will reject subsequent connection attempts until the number of active client sessions drops below the specified threshold. Note that adjusting these values lower than the current connection limits will not affect clients that have already connected to the server. For example, if the **Start** method is called with the maximum number of clients set to 100, and then the **Throttle** method is called lowering that value to 75, no existing client connections will be affected by the change. However, the server will not accept any new connections until the number of active clients drops below 75.

If the value of the *maxGuests* parameter is greater than zero, then anonymous logins will be enabled and clients can authenticate with the username "anonymous" and their email address as the password. If the parameter is set to zero, then anonymous logins will be disabled. Note that this will not affect any clients that are currently logged in, it only affects those clients that connect after the **Throttle** method has been called.

Increasing the connection rate value will force the server to slow down the rate at which it will accept incoming client connection requests. For example, setting this parameter to a value of 1000 would limit the server to accepting one client connection every second, while a value of 250 would allow the server to accept four client connections per second. Note that significantly increasing the amount of time the server must wait to accept client connections can exceed the connection backlog queue, resulting in client connections being rejected.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.Throttle Overload List

FtpServer.Throttle Method (Int32, Int32)

Limit the maximum number of client connections and connections per IP address.

[Visual Basic]

```
Overloads Public Function Throttle( _
   ByVal maxClients As Integer, _
   ByVal maxClientsPerAddress As Integer _
) As Boolean
```

[C#]

```
public bool Throttle(
   int maxClients,
   int maxClientsPerAddress
);
```

Parameters

maxClients

An integer value that specifies the maximum number of clients that may connect to the server. A value of zero specifies that there is no fixed limit to the number of client connections.

maxClientsPerAddress

An integer value that specifies the maximum number of clients that may connect to the server from the same IP address. A value of zero specifies that there is no fixed limit to the number of client connections per address. By default, there is no limit on the number of client connections per address.

Return Value

A boolean value which specifies if the method was successful. A return value of **true** indicates success. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

If the maximum number of client connections or maximum number of connections per address is exceeded, the server will reject subsequent connection attempts until the number of active client sessions drops below the specified threshold. Note that adjusting these values lower than the current connection limits will not affect clients that have already connected to the server. For example, if the **Start** method is called with the maximum number of clients set to 100, and then the **Throttle** method is called lowering that value to 75, no existing client connections will be affected by the change. However, the server will not accept any new connections until the number of active clients drops below 75.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.Throttle Overload List

FtpServer.Throttle Method (Int32)

Limit the maximum number of client connections.

[Visual Basic]

```
Overloads Public Function Throttle( _
   ByVal maxClients As Integer _
) As Boolean
```

[C#]

```
public bool Throttle(
   int maxClients
);
```

Parameters

maxClients

An integer value that specifies the maximum number of clients that may connect to the server. A value of zero specifies that there is no fixed limit to the number of client connections.

Return Value

A boolean value which specifies if the method was successful. A return value of **true** indicates success. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

If the maximum number of client connections or maximum number of connections per address is exceeded, the server will reject subsequent connection attempts until the number of active client sessions drops below the specified threshold. Note that adjusting these values lower than the current connection limits will not affect clients that have already connected to the server. For example, if the **Start** method is called with the maximum number of clients set to 100, and then the **Throttle** method is called lowering that value to 75, no existing client connections will be affected by the change. However, the server will not accept any new connections until the number of active clients drops below 75.

See Also

FtpServer Class | SocketTools Namespace | FtpServer.Throttle Overload List

FtpServer.Uninitialize Method

Uninitialize the class library and release any resources allocated for the server.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method terminates any active connection, releases resources allocated for the server and unloads the networking library. After this method has been called, no further network operations may be performed until the class instance has been re-initialized.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the class is no longer needed.

See Also

FtpServer Class | SocketTools Namespace | Initialize Method

FtpServer Events

The events of the **FtpServer** class are listed below. For a complete list of **FtpServer** class members, see the FtpServer Members topic.

Public Instance Events

⚡ OnAuthenticate	Occurs when the client has requested authentication with the specified username and password.
⚡ OnCommand	Occurs when a client has issued a command to the server.
⚡ OnConnect	Occurs when a connection is established with the remote host.
⚡ OnDisconnect	Occurs when the remote host disconnects from the local system.
⚡ OnDownload	Occurs when a connection is established with the remote host.
⚡ OnError	Occurs when an network operation fails.
⚡ OnExecute	Occurs when the client has executed an external program on the server.
⚡ OnIdle	Occurs when the there are no clients connected to the server.
⚡ OnLogin	Occurs when the client has successfully authenticated the session.
⚡ OnLogout	Occurs when the client has logged out or reinitialized the session.
⚡ OnResult	Occurs when the command issued by the client has been processed by the server.
⚡ OnStart	Occurs when the server starts accepting connections.
⚡ OnStop	Occurs when the server stops accepting connections.
⚡ OnTimeout	Occurs when the client has exceeded the maximum allowed idle time.
⚡ OnUpload	Occurs when the client has successfully uploaded a file to the server.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.OnAuthenticate Event

Occurs when the client has requested authentication with the specified username and password.

[Visual Basic]
```
Public Event OnAuthenticate As OnAuthenticateEventHandler
```

[C#]
```
public event OnAuthenticateEventHandler OnAuthenticate;
```

Event Data

The event handler receives an argument of type FtpServer.AuthenticateEventArgs containing data related to this event. The following **FtpServer.AuthenticateEventArgs** properties provide information specific to this event.

Property	Description
ClientId	Gets a value that uniquely identifies the client session.
HostName	Gets the host name used by the client to establish the connection to the server.
Password	Gets the password provided by the client for authentication.
UserName	Gets the password provided by the client for authentication.

Remarks

The **OnAuthenticate** event occurs when the client has requested authentication by sending the USER and PASS command to the server. The event handler can call the **Authenticate** method to authenticate the client session. If the client is not authenticated, the server will send an error message to the client and terminate the session.

If the application has created one or more virtual users using the **AddUser** method and/or the **LocalUser** property has been set to **True**, it is not necessary to implement an **OnAuthenticate** handler unless you also wish to perform custom authentication for specific users.

This event handler will be invoked in the context of the worker thread that is managing the client session, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.AuthenticateEventArgs Class

Provides data for the OnAuthenticate event.

For a list of all members of this type, see FtpServer.AuthenticateEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FtpServer.AuthenticateEventArgs

[Visual Basic]

```
Public Class FtpServer.AuthenticateEventArgs
    Inherits EventArgs
```

[C#]

```
public class FtpServer.AuthenticateEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

FtpServer.AuthenticateEventArgs Members | SocketTools Namespace

FtpServer.AuthenticateEventArgs Constructor

Initializes a new instance of the FtpServer.AuthenticateEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FtpServer.AuthenticateEventArgs();
```

See Also

FtpServer.AuthenticateEventArgs Class | SocketTools Namespace

FtpServer.AuthenticateEventArgs Members

FtpServer.AuthenticateEventArgs overview

Public Instance Constructors

FtpServer.AuthenticateEventArgs Constructor	Initializes a new instance of the FtpServer.AuthenticateEventArgs class.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
HostName	Gets the host name used by the client to establish the connection to the server.
Password	Gets the password provided by the client for authentication.
UserName	Gets the password provided by the client for authentication.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpServer.AuthenticateEventArgs Class | SocketTools Namespace

FtpServer.AuthenticateEventArgs Properties

The properties of the **FtpServer.AuthenticateEventArgs** class are listed below. For a complete list of **FtpServer.AuthenticateEventArgs** class members, see the FtpServer.AuthenticateEventArgs Members topic.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
HostName	Gets the host name used by the client to establish the connection to the server.
Password	Gets the password provided by the client for authentication.
UserName	Gets the password provided by the client for authentication.

See Also

FtpServer.AuthenticateEventArgs Class | SocketTools Namespace

FtpServer.AuthenticateEventArgs.ClientId Property

Gets a value that uniquely identifies the client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which identifies the client session.

Remarks

The **ClientId** property returns a unique integer value that identifies the client session that generated the event.

See Also

FtpServer.AuthenticateEventArgs Class | SocketTools Namespace

FtpServer.AuthenticateEventArgs.HostName Property

Gets the host name used by the client to establish the connection to the server.

[Visual Basic]

```
Public ReadOnly Property HostName As String
```

[C#]

```
public string HostName {get;}
```

Property Value

A string that specifies a host name.

See Also

FtpServer.AuthenticateEventArgs Class | SocketTools Namespace

FtpServer.AuthenticateEventArgs.Password Property

Gets the password provided by the client for authentication.

[Visual Basic]
```
Public ReadOnly Property Password As String
```

[C#]
```
public string Password {get;}
```

Property Value

A string that specifies the password provided by the client when it requests authentication.

See Also

FtpServer.AuthenticateEventArgs Class | SocketTools Namespace

FtpServer.AuthenticateEventArgs.UserName Property

Gets the password provided by the client for authentication.

[Visual Basic]

```
Public ReadOnly Property UserName As String
```

[C#]

```
public string UserName {get;}
```

Property Value

A string that specifies the username provided by the client when it requests authentication.

See Also

FtpServer.AuthenticateEventArgs Class | SocketTools Namespace

FtpServer.OnCommand Event

Occurs when a client has issued a command to the server.

[Visual Basic]
```
Public Event OnCommand As OnCommandEventHandler
```

[C#]
```
public event OnCommandEventHandler OnCommand;
```

Event Data

The event handler receives an argument of type FtpServer.CommandEventArgs containing data related to this event. The following **FtpServer.CommandEventArgs** properties provide information specific to this event.

Property	Description
ClientId	Gets a value that uniquely identifies the client session.
Command	Gets a value that specifies the command issued by the client.
Parameters	Gets a value that specifies the command parameters issued by the client.

Remarks

The **OnCommand** event occurs after the client has sent a command to the server, but before the command has been processed. This event occurs for all commands issued by the client, including invalid or disabled commands. If the application wishes to handle the command itself, it must perform any processing and then call the **SendResponse** method to send the success or error code to the client. If the **SendResponse** method is not called, then the server will perform its default processing for the command.

After the command has been processed, the **OnResult** event handler will be invoked.

This event handler will be invoked in the context of the worker thread that is managing the client session, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.CommandEventArgs Class

Provides data for the OnCommand event.

For a list of all members of this type, see FtpServer.CommandEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FtpServer.CommandEventArgs

[Visual Basic]
```
Public Class FtpServer.CommandEventArgs
    Inherits EventArgs
```

[C#]
```
public class FtpServer.CommandEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

FtpServer.CommandEventArgs Members | SocketTools Namespace

FtpServer.CommandEventArgs Constructor

Initializes a new instance of the FtpServer.CommandEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FtpServer.CommandEventArgs();
```

See Also

FtpServer.CommandEventArgs Class | SocketTools Namespace

FtpServer.CommandEventArgs Members

FtpServer.CommandEventArgs overview

Public Instance Constructors

FtpServer.CommandEventArgs Constructor	Initializes a new instance of the FtpServer.CommandEventArgs class.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
Command	Gets a value that specifies the command issued by the client.
Parameters	Gets a value that specifies the command parameters issued by the client.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpServer.CommandEventArgs Class | SocketTools Namespace

FtpServer.CommandEventArgs Properties

The properties of the **FtpServer.CommandEventArgs** class are listed below. For a complete list of **FtpServer.CommandEventArgs** class members, see the FtpServer.CommandEventArgs Members topic.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
Command	Gets a value that specifies the command issued by the client.
Parameters	Gets a value that specifies the command parameters issued by the client.

See Also

FtpServer.CommandEventArgs Class | SocketTools Namespace

FtpServer.CommandEventArgs.ClientId Property

Gets a value that uniquely identifies the client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which identifies the client session.

Remarks

The **ClientId** property returns a unique integer value that identifies the client session that generated the event.

See Also

FtpServer.CommandEventArgs Class | SocketTools Namespace

FtpServer.CommandEventArgs.Command Property

Gets a value that specifies the command issued by the client.

[Visual Basic]

```
Public ReadOnly Property Command As String
```

[C#]

```
public string Command {get;}
```

Property Value

A string that specifies the command sent by the client.

See Also

FtpServer.CommandEventArgs Class | SocketTools Namespace

FtpServer.CommandEventArgs.Parameters Property

Gets a value that specifies the command parameters issued by the client.

[Visual Basic]

```
Public ReadOnly Property Parameters As String
```

[C#]

```
public string Parameters {get;}
```

Property Value

A string that specifies the command parameters sent by the client.

See Also

FtpServer.CommandEventArgs Class | SocketTools Namespace

FtpServer.OnConnect Event

Occurs when a connection is established with the remote host.

[Visual Basic]

```
Public Event OnConnect As OnConnectEventHandler
```

[C#]

```
public event OnConnectEventHandler OnConnect;
```

Event Data

The event handler receives an argument of type FtpServer.ConnectEventArgs containing data related to this event. The following **FtpServer.ConnectEventArgs** properties provide information specific to this event.

Property	Description
ClientAddress	Gets the address of the client establishing the connection.
ClientId	Gets a value that uniquely identifies the client session.

Remarks

The **OnConnect** event occurs after the client has established its initial connection to the server, after the server has checked the active client limits and the TLS handshake has been performed if required. If the server has been suspended, or the limit on the maximum number of client sessions has been exceeded, the server will terminate the client session prior to this event handler being invoked.

If no event handler is implemented, the server will perform the default action of accepting the connection and sending a standard greeting to the client. If you want your application to send a custom greeting to the client when it connects, call the **SendResponse** method, specifying a result code of 220 and a message of your choice.

To reject a connection, call the **SendResponse** method to send an error response to the client. Typically the result code value would be 421 to indicate that the server will not accept the connection. Next, call the **DisconnectClient** method to terminate the client session.

This event handler will be invoked in the context of the worker thread that is managing the client session, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.ConnectEventArgs Class

Provides data for the OnConnect event.

For a list of all members of this type, see FtpServer.ConnectEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FtpServer.ConnectEventArgs

[Visual Basic]
```
Public Class FtpServer.ConnectEventArgs
    Inherits EventArgs
```

[C#]
```
public class FtpServer.ConnectEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

FtpServer.ConnectEventArgs Members | SocketTools Namespace | OnConnect Event (SocketTools.FtpServer)

FtpServer.ConnectEventArgs Constructor

Initializes a new instance of the FtpServer.ConnectEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FtpServer.ConnectEventArgs();
```

See Also

FtpServer.ConnectEventArgs Class | SocketTools Namespace

FtpServer.ConnectEventArgs Members

FtpServer.ConnectEventArgs overview

Public Instance Constructors

FtpServer.ConnectEventArgs Constructor	Initializes a new instance of the FtpServer.ConnectEventArgs class.

Public Instance Properties

ClientAddress	Gets the address of the client establishing the connection.
ClientId	Gets a value that uniquely identifies the client session.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpServer.ConnectEventArgs Class | SocketTools Namespace | OnConnect Event (SocketTools.FtpServer)

FtpServer.ConnectEventArgs Properties

The properties of the **FtpServer.ConnectEventArgs** class are listed below. For a complete list of **FtpServer.ConnectEventArgs** class members, see the FtpServer.ConnectEventArgs Members topic.

Public Instance Properties

ClientAddress	Gets the address of the client establishing the connection.
ClientId	Gets a value that uniquely identifies the client session.

See Also

FtpServer.ConnectEventArgs Class | SocketTools Namespace | OnConnect Event (SocketTools.FtpServer)

FtpServer.ConnectEventArgs.ClientAddress Property

Gets the address of the client establishing the connection.

[Visual Basic]

```
Public ReadOnly Property ClientAddress As String
```

[C#]

```
public string ClientAddress {get;}
```

Property Value

A string that specifies the Internet Protocol address of the client.

See Also

FtpServer.ConnectEventArgs Class | SocketTools Namespace

FtpServer.ConnectEventArgs.ClientId Property

Gets a value that uniquely identifies the client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which identifies the client session.

Remarks

The **ClientId** property returns a unique integer value that identifies the client session that generated the event.

See Also

FtpServer.ConnectEventArgs Class | SocketTools Namespace

FtpServer.OnDownload Event

Occurs when a connection is established with the remote host.

[Visual Basic]
```
Public Event OnDownload As OnDownloadEventHandler
```

[C#]
```
public event OnDownloadEventHandler OnDownload;
```

Event Data

The event handler receives an argument of type FtpServer.DownloadEventArgs containing data related to this event. The following **FtpServer.DownloadEventArgs** properties provide information specific to this event.

Property	Description
ClientId	Gets a value that uniquely identifies the client session.
FileName	Gets a value that specifies the file being downloaded.
FileSize	Gets a value that specifies the size of the file.

Remarks

The **OnDownload** event occurs after the client has successfully downloaded a file from the server using the RETR command. If the file transfer fails or is aborted, this event will not occur.

This event handler will be invoked in the context of the worker thread that is managing the client session, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.DownloadEventArgs Class

Provides data for the OnDownload event.

For a list of all members of this type, see FtpServer.DownloadEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FtpServer.DownloadEventArgs

[Visual Basic]
```
Public Class FtpServer.DownloadEventArgs
    Inherits EventArgs
```

[C#]
```
public class FtpServer.DownloadEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

FtpServer.DownloadEventArgs Members | SocketTools Namespace

FtpServer.DownloadEventArgs Constructor

Initializes a new instance of the FtpServer.DownloadEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public FtpServer.DownloadEventArgs();
```

See Also

FtpServer.DownloadEventArgs Class | SocketTools Namespace

FtpServer.DownloadEventArgs Members

FtpServer.DownloadEventArgs overview

Public Instance Constructors

FtpServer.DownloadEventArgs Constructor	Initializes a new instance of the FtpServer.DownloadEventArgs class.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
FileName	Gets a value that specifies the file being downloaded.
FileSize	Gets a value that specifies the size of the file.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpServer.DownloadEventArgs Class | SocketTools Namespace

FtpServer.DownloadEventArgs Properties

The properties of the **FtpServer.DownloadEventArgs** class are listed below. For a complete list of **FtpServer.DownloadEventArgs** class members, see the FtpServer.DownloadEventArgs Members topic.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
FileName	Gets a value that specifies the file being downloaded.
FileSize	Gets a value that specifies the size of the file.

See Also

FtpServer.DownloadEventArgs Class | SocketTools Namespace

FtpServer.DownloadEventArgs.ClientId Property

Gets a value that uniquely identifies the client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which identifies the client session.

Remarks

The **ClientId** property returns a unique integer value that identifies the client session that generated the event.

See Also

FtpServer.DownloadEventArgs Class | SocketTools Namespace

FtpServer.DownloadEventArgs.FileName Property

Gets a value that specifies the file being downloaded.

[Visual Basic]

```
Public ReadOnly Property FileName As String
```

[C#]

```
public string FileName {get;}
```

Property Value

A string that specifies the full path to a file on the local system.

See Also

FtpServer.DownloadEventArgs Class | SocketTools Namespace

FtpServer.DownloadEventArgs.FileSize Property

Gets a value that specifies the size of the file.

[Visual Basic]

```
Public ReadOnly Property FileSize As Long
```

[C#]

```
public long FileSize {get;}
```

Property Value

A long integer value that specifies the size of the file in bytes.

See Also

FtpServer.DownloadEventArgs Class | SocketTools Namespace

FtpServer.OnDisconnect Event

Occurs when the remote host disconnects from the local system.

[Visual Basic]

```
Public Event OnDisconnect As OnDisconnectEventHandler
```

[C#]

```
public event OnDisconnectEventHandler OnDisconnect;
```

Event Data

The event handler receives an argument of type FtpServer.DisconnectEventArgs containing data related to this event. The following **FtpServer.DisconnectEventArgs** property provides information specific to this event.

Property	Description
ClientId	Gets a value that uniquely identifies the client session.

Remarks

The **OnDisconnect** event is generated when the connection is terminated by the client and there is no more data available to be read.

It is not necessary to call the **Disconnect** method inside the **OnDisconnect** event handler because the client session is already in the process of disconnecting from the server.

This event handler will be invoked in the context of the worker thread that is managing the client session, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.DisconnectEventArgs Class

Provides data for the OnDisconnect event.

For a list of all members of this type, see FtpServer.DisconnectEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FtpServer.DisconnectEventArgs

[Visual Basic]
```
Public Class FtpServer.DisconnectEventArgs
    Inherits EventArgs
```

[C#]
```
public class FtpServer.DisconnectEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

FtpServer.DisconnectEventArgs Members | SocketTools Namespace | OnDisconnect Event (SocketTools.FtpServer)

FtpServer.DisconnectEventArgs Constructor

Initializes a new instance of the FtpServer.DisconnectEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FtpServer.DisconnectEventArgs();
```

See Also

FtpServer.DisconnectEventArgs Class | SocketTools Namespace

FtpServer.DisconnectEventArgs Members

FtpServer.DisconnectEventArgs overview

Public Instance Constructors

FtpServer.DisconnectEventArgs Constructor	Initializes a new instance of the FtpServer.DisconnectEventArgs class.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpServer.DisconnectEventArgs Class | SocketTools Namespace | OnDisconnect Event (SocketTools.FtpServer)

FtpServer.DisconnectEventArgs Properties

The properties of the **FtpServer.DisconnectEventArgs** class are listed below. For a complete list of **FtpServer.DisconnectEventArgs** class members, see the FtpServer.DisconnectEventArgs Members topic.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.

See Also

FtpServer.DisconnectEventArgs Class | SocketTools Namespace | OnDisconnect Event (SocketTools.FtpServer)

FtpServer.DisconnectEventArgs.ClientId Property

Gets a value that uniquely identifies the client session.

[Visual Basic]
```
Public ReadOnly Property ClientId As Integer
```

[C#]
```
public int ClientId {get;}
```

Property Value

An integer value which identifies the client session.

Remarks

The **ClientId** property returns a unique integer value that identifies the client session that generated the event.

See Also

FtpServer.DisconnectEventArgs Class | SocketTools Namespace

FtpServer.OnError Event

Occurs when an network operation fails.

[Visual Basic]
```
Public Event OnError As OnErrorEventHandler
```

[C#]
```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type FtpServer.ErrorEventArgs containing data related to this event. The following **FtpServer.ErrorEventArgs** properties provide information specific to this event.

Property	Description
ClientId	Gets a value that uniquely identifies the client session.
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a network operation fails.

This event handler may be invoked in the context of the worker thread that is managing the client session, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see FtpServer.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FtpServer.ErrorEventArgs

[Visual Basic]

```
Public Class FtpServer.ErrorEventArgs
    Inherits EventArgs
```

[C#]

```
public class FtpServer.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

FtpServer.ErrorEventArgs Members | SocketTools Namespace

FtpServer.ErrorEventArgs Constructor

Initializes a new instance of the FtpServer.ErrorEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FtpServer.ErrorEventArgs();
```

See Also

FtpServer.ErrorEventArgs Class | SocketTools Namespace

FtpServer.ErrorEventArgs Members

FtpServer.ErrorEventArgs overview

Public Instance Constructors

FtpServer.ErrorEventArgs Constructor	Initializes a new instance of the FtpServer.ErrorEventArgs class.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpServer.ErrorEventArgs Class | SocketTools Namespace

FtpServer.ErrorEventArgs Properties

The properties of the **FtpServer.ErrorEventArgs** class are listed below. For a complete list of **FtpServer.ErrorEventArgs** class members, see the FtpServer.ErrorEventArgs Members topic.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

FtpServer.ErrorEventArgs Class | SocketTools Namespace

FtpServer.ErrorEventArgs.ClientId Property

Gets a value that uniquely identifies the client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which identifies the client session.

Remarks

The **ClientId** property returns a unique integer value that identifies the client session that generated the event.

See Also

FtpServer.ErrorEventArgs Class | SocketTools Namespace

FtpServer.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]
```
Public ReadOnly Property Description As String
```

[C#]
```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

FtpServer.ErrorEventArgs Class | SocketTools Namespace | Error Property

FtpServer.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public FtpServer.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

FtpServer.ErrorEventArgs Class | SocketTools Namespace | Description Property

FtpServer.OnExecute Event

Occurs when the client has executed an external program on the server.

[Visual Basic]
```
Public Event OnExecute As OnExecuteEventHandler
```

[C#]
```
public event OnExecuteEventHandler OnExecute;
```

Event Data

The event handler receives an argument of type FtpServer.ExecuteEventArgs containing data related to this event. The following **FtpServer.ExecuteEventArgs** properties provide information specific to this event.

Property	Description
ClientId	Gets a value that uniquely identifies the client session.
ExitCode	Gets a value that specifies the exit code for the external program.
Output	Gets the output of the external command executed by the client.
Program	Gets the name of the external program executed by the client.

Remarks

The **OnExecute** event occurs after the client has successfully executed an external program using the SITE EXEC command.

This event will only be generated if the client has the **ftpAccessExecute** permission. Clients are not granted this permission by default, and must be explicitly permitted to execute external programs. If the client does have this permission, it can only execute specific programs that have been registered by the server application using the **RegisterProgram** method.

This event handler will be invoked in the context of the worker thread that is managing the client session, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.ExecuteEventArgs Class

Provides data for the OnExecute event.

For a list of all members of this type, see FtpServer.ExecuteEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FtpServer.ExecuteEventArgs

[Visual Basic]

```
Public Class FtpServer.ExecuteEventArgs
    Inherits EventArgs
```

[C#]

```
public class FtpServer.ExecuteEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

FtpServer.ExecuteEventArgs Members | SocketTools Namespace

FtpServer.ExecuteEventArgs Constructor

Initializes a new instance of the FtpServer.ExecuteEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FtpServer.ExecuteEventArgs();
```

See Also

FtpServer.ExecuteEventArgs Class | SocketTools Namespace

FtpServer.ExecuteEventArgs Members

FtpServer.ExecuteEventArgs overview

Public Instance Constructors

FtpServer.ExecuteEventArgs Constructor	Initializes a new instance of the FtpServer.ExecuteEventArgs class.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
ExitCode	Gets a value that specifies the exit code for the external program.
Output	Gets the output of the external command executed by the client.
Program	Gets the name of the external program executed by the client.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpServer.ExecuteEventArgs Class | SocketTools Namespace

FtpServer.ExecuteEventArgs Properties

The properties of the **FtpServer.ExecuteEventArgs** class are listed below. For a complete list of **FtpServer.ExecuteEventArgs** class members, see the FtpServer.ExecuteEventArgs Members topic.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
ExitCode	Gets a value that specifies the exit code for the external program.
Output	Gets the output of the external command executed by the client.
Program	Gets the name of the external program executed by the client.

See Also

FtpServer.ExecuteEventArgs Class | SocketTools Namespace

FtpServer.ExecuteEventArgs.ClientId Property

Gets a value that uniquely identifies the client session.

[Visual Basic]
```
Public ReadOnly Property ClientId As Integer
```

[C#]
```
public int ClientId {get;}
```

Property Value

An integer value which identifies the client session.

Remarks

The **ClientId** property returns a unique integer value that identifies the client session that generated the event.

See Also

FtpServer.ExecuteEventArgs Class | SocketTools Namespace

FtpServer.ExecuteEventArgs.ExitCode Property

Gets a value that specifies the exit code for the external program.

[Visual Basic]

```
Public ReadOnly Property ExitCode As Integer
```

[C#]

```
public int ExitCode {get;}
```

Property Value

An integer value that specifies an exit code.

See Also

FtpServer.ExecuteEventArgs Class | SocketTools Namespace

FtpServer.ExecuteEventArgs.Output Property

Gets the output of the external command executed by the client.

[Visual Basic]

```
Public ReadOnly Property Output As String
```

[C#]

```
public string Output {get;}
```

Property Value

A string that contains the output of the external program executed by the client.

See Also

FtpServer.ExecuteEventArgs Class | SocketTools Namespace

FtpServer.ExecuteEventArgs.Program Property

Gets the name of the external program executed by the client.

[Visual Basic]

```
Public ReadOnly Property Program As String
```

[C#]

```
public string Program {get;}
```

Property Value

A string that identifies the external program executed by the client.

See Also

FtpServer.ExecuteEventArgs Class | SocketTools Namespace

FtpServer.OnIdle Event

Occurs when the there are no clients connected to the server.

[Visual Basic]

```
Public Event OnIdle As EventHandler
```

[C#]

```
public event EventHandler OnIdle;
```

Remarks

This event will only occur after at least one client has connected to the server and then closes its connection or is disconnected. This event will not occur immediately after the server has started using the **Start** method, and will not occur when the server is stopped using the **Stop** method. Your application should implement an **OnStart** event handler for when the server first starts, and an **OnStop** event handler for when the server is stopped.

If one or more new client connections are accepted after this event occurs, the event will be generated again when those clients disconnect and the active client count drops to zero. Therefore it is to be expected that this event will occur multiple times over the lifetime of the server as it continues to listen for connections

This event handler will be invoked in the context of the worker thread that is managing the server, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.OnLogin Event

Occurs when the client has successfully authenticated the session.

[Visual Basic]
```
Public Event OnLogin As OnLoginEventHandler
```

[C#]
```
public event OnLoginEventHandler OnLogin;
```

Event Data

The event handler receives an argument of type FtpServer.LoginEventArgs containing data related to this event. The following **FtpServer.LoginEventArgs** properties provide information specific to this event.

Property	Description
ClientId	Gets a value that uniquely identifies the client session.
Directory	Gets the home directory of the user that has logged into the server.
UserName	Gets the username associated with the client that logged into the server.

Remarks

The **OnLogin** event occurs after the client has successfully authenticated itself using the USER and PASS commands.

This event handler will be invoked in the context of the worker thread that is managing the client session, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.LoginEventArgs Class

Provides data for the OnLogin event.

For a list of all members of this type, see FtpServer.LoginEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FtpServer.LoginEventArgs

[Visual Basic]
```
Public Class FtpServer.LoginEventArgs
    Inherits EventArgs
```

[C#]
```
public class FtpServer.LoginEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

FtpServer.LoginEventArgs Members | SocketTools Namespace

FtpServer.LoginEventArgs Constructor

Initializes a new instance of the FtpServer.LoginEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public FtpServer.LoginEventArgs();
```

See Also

FtpServer.LoginEventArgs Class | SocketTools Namespace

FtpServer.LoginEventArgs Members

FtpServer.LoginEventArgs overview

Public Instance Constructors

FtpServer.LoginEventArgs Constructor	Initializes a new instance of the FtpServer.LoginEventArgs class.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
Directory	Gets the home directory of the user that has logged into the server.
UserName	Gets the username associated with the client that logged into the server.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpServer.LoginEventArgs Class | SocketTools Namespace

FtpServer.LoginEventArgs Properties

The properties of the **FtpServer.LoginEventArgs** class are listed below. For a complete list of **FtpServer.LoginEventArgs** class members, see the FtpServer.LoginEventArgs Members topic.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
Directory	Gets the home directory of the user that has logged into the server.
UserName	Gets the username associated with the client that logged into the server.

See Also

FtpServer.LoginEventArgs Class | SocketTools Namespace

FtpServer.LoginEventArgs.ClientId Property

Gets a value that uniquely identifies the client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which identifies the client session.

Remarks

The **ClientId** property returns a unique integer value that identifies the client session that generated the event.

See Also

FtpServer.LoginEventArgs Class | SocketTools Namespace

FtpServer.LoginEventArgs.Directory Property

Gets the home directory of the user that has logged into the server.

[Visual Basic]

```
Public ReadOnly Property Directory As String
```

[C#]

```
public string Directory {get;}
```

Property Value

A string that specifies the full path to a local directory on the server.

See Also

FtpServer.LoginEventArgs Class | SocketTools Namespace

FtpServer.LoginEventArgs.UserName Property

Gets the username associated with the client that logged into the server.

[Visual Basic]

```
Public ReadOnly Property UserName As String
```

[C#]

```
public string UserName {get;}
```

Property Value

A string that specifies the username associated with the client session.

See Also

FtpServer.LoginEventArgs Class | SocketTools Namespace

FtpServer.OnLogout Event

Occurs when the client has logged out or reinitialized the session.

[Visual Basic]

```
Public Event OnLogout As OnLogoutEventHandler
```

[C#]

```
public event OnLogoutEventHandler OnLogout;
```

Event Data

The event handler receives an argument of type FtpServer.LogoutEventArgs containing data related to this event. The following **FtpServer.LogoutEventArgs** properties provide information specific to this event.

Property	Description
ClientId	Gets a value that uniquely identifies the client session.
UserName	Gets the username associated with the client that logged out from the server.

Remarks

The **OnLogout** event occurs after the client has successfully logged out using the QUIT command or reinitialized the session using the REIN command.

The application should not depend on this event handler always being invoked when a client is disconnected from the server. This event only occurs when the client sends the QUIT or REIN commands and will not be invoked if the client connection is aborted or disconnected for some other reason, such as exceeding the idle timeout period. If the application needs to update data structures or perform some cleanup when a client disconnects, that should be done in the **OnDisconnect** event handler.

The application should not call the **Disconnect** method in the handler for this event because the client is either in the process of disconnecting or expects that it can submit new credentials to the server.

This event handler will be invoked in the context of the worker thread that is managing the client session, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.LogoutEventArgs Class

Provides data for the OnLogout event.

For a list of all members of this type, see FtpServer.LogoutEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FtpServer.LogoutEventArgs

[Visual Basic]
```
Public Class FtpServer.LogoutEventArgs
    Inherits EventArgs
```

[C#]
```
public class FtpServer.LogoutEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

FtpServer.LogoutEventArgs Members | SocketTools Namespace

FtpServer.LogoutEventArgs Constructor

Initializes a new instance of the FtpServer.LogoutEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public FtpServer.LogoutEventArgs();
```

See Also

FtpServer.LogoutEventArgs Class | SocketTools Namespace

FtpServer.LogoutEventArgs Members

FtpServer.LogoutEventArgs overview

Public Instance Constructors

FtpServer.LogoutEventArgs Constructor	Initializes a new instance of the FtpServer.LogoutEventArgs class.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
UserName	Gets the username associated with the client that logged out from the server.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpServer.LogoutEventArgs Class | SocketTools Namespace

FtpServer.LogoutEventArgs Properties

The properties of the **FtpServer.LogoutEventArgs** class are listed below. For a complete list of **FtpServer.LogoutEventArgs** class members, see the FtpServer.LogoutEventArgs Members topic.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
UserName	Gets the username associated with the client that logged out from the server.

See Also

FtpServer.LogoutEventArgs Class | SocketTools Namespace

FtpServer.LogoutEventArgs.ClientId Property

Gets a value that uniquely identifies the client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which identifies the client session.

Remarks

The **ClientId** property returns a unique integer value that identifies the client session that generated the event.

See Also

FtpServer.LogoutEventArgs Class | SocketTools Namespace

FtpServer.LogoutEventArgs.UserName Property

Gets the username associated with the client that logged out from the server.

[Visual Basic]

```
Public ReadOnly Property UserName As String
```

[C#]

```
public string UserName {get;}
```

Property Value

A string that specifies the username associated with the client session.

See Also

FtpServer.LogoutEventArgs Class | SocketTools Namespace

FtpServer.OnResult Event

Occurs when the command issued by the client has been processed by the server.

[Visual Basic]
```
Public Event OnResult As OnResultEventHandler
```

[C#]
```
public event OnResultEventHandler OnResult;
```

Event Data

The event handler receives an argument of type FtpServer.ResultEventArgs containing data related to this event. The following **FtpServer.ResultEventArgs** properties provide information specific to this event.

Property	Description
ClientId	Gets a value that uniquely identifies the client session.
Command	Gets a value that specifies the command issued by the client.
ResultCode	Gets the result code associated with the last command issued by the client.

Remarks

The **OnResult** event occurs after the server has processed a command issued by the client. This event will inform the application whether the command that was issued by the client was successful or not. If the command was successful, then other related events such as **OnDownload** may also fire after this event.

This event handler will be invoked in the context of the worker thread that is managing the client session, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.ResultEventArgs Class

Provides data for the OnResult event.

For a list of all members of this type, see FtpServer.ResultEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FtpServer.ResultEventArgs

[Visual Basic]

```
Public Class FtpServer.ResultEventArgs
    Inherits EventArgs
```

[C#]

```
public class FtpServer.ResultEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

FtpServer.ResultEventArgs Members | SocketTools Namespace

FtpServer.ResultEventArgs Constructor

Initializes a new instance of the FtpServer.ResultEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public FtpServer.ResultEventArgs();
```

See Also

FtpServer.ResultEventArgs Class | SocketTools Namespace

FtpServer.ResultEventArgs Members

FtpServer.ResultEventArgs overview

Public Instance Constructors

FtpServer.ResultEventArgs Constructor	Initializes a new instance of the FtpServer.ResultEventArgs class.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
Command	Gets a value that specifies the command issued by the client.
ResultCode	Gets the result code associated with the last command issued by the client.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpServer.ResultEventArgs Class | SocketTools Namespace

FtpServer.ResultEventArgs Properties

The properties of the **FtpServer.ResultEventArgs** class are listed below. For a complete list of **FtpServer.ResultEventArgs** class members, see the FtpServer.ResultEventArgs Members topic.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
Command	Gets a value that specifies the command issued by the client.
ResultCode	Gets the result code associated with the last command issued by the client.

See Also

FtpServer.ResultEventArgs Class | SocketTools Namespace

FtpServer.ResultEventArgs.ClientId Property

Gets a value that uniquely identifies the client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which identifies the client session.

Remarks

The **ClientId** property returns a unique integer value that identifies the client session that generated the event.

See Also

FtpServer.ResultEventArgs Class | SocketTools Namespace

FtpServer.ResultEventArgs.Command Property

Gets a value that specifies the command issued by the client.

[Visual Basic]

```
Public ReadOnly Property Command As String
```

[C#]

```
public string Command {get;}
```

Property Value

A string that specifies the command sent by the client.

See Also

FtpServer.ResultEventArgs Class | SocketTools Namespace

FtpServer.ResultEventArgs.ResultCode Property

Gets the result code associated with the last command issued by the client.

[Visual Basic]

```
Public ReadOnly Property ResultCode As Integer
```

[C#]

```
public int ResultCode {get;}
```

Property Value

An integer value that indicates if the command completed successfully.

Remarks

The **ResultCode** property is a three-digit numeric code that is used to indicate success or failure. These codes are defined as part of the File Transfer Protocol standard, with values in the range of 200-299 indicating success. Values in the range of 400-499 and 500-599 indicate failure due to various error conditions. Examples of such failures would be attempting to access a file that does not exist, issuing an unrecognized command or attempting to perform a privileged operation

See Also

FtpServer.ResultEventArgs Class | SocketTools Namespace

FtpServer.OnStart Event

Occurs when the server starts accepting connections.

[Visual Basic]

```
Public Event OnStart As EventHandler
```

[C#]

```
public event EventHandler OnStart;
```

Remarks

The **OnStart** event occurs after the **Start** method has been called and the server and begins listening for connections from clients. An application can use this event to update the user interface and perform any additional initialization functions that are required by the application

See Also

FtpServer Class | SocketTools Namespace

FtpServer.OnStop Event

Occurs when the server stops accepting connections.

[Visual Basic]
```
Public Event OnStop As EventHandler
```

[C#]
```
public event EventHandler OnStop;
```

Remarks

The **OnStop** event occurs after the **Stop** method has been called and all active client sessions have terminated. An application can use this event to update the user interface and perform any additional cleanup functions that are required by the application. If the server has a large number of active clients, this event may not occur immediately. The **OnDisconnect** event will fire for each client as the server is in the process of shutting down. During the shutdown process, the server is still considered to be active, however it will not accept any further connections. When the **OnStop** event is fired, the server thread has terminated and the listening socket has been closed.

This event will not occur if the server is forcibly stopped using the **Reset** method, or when the **Uninitialize** method is called prior to disposing an instance of the control. Applications that depend on this event should ensure that the server is shutdown gracefully using the **Stop** method prior to terminating the application

See Also

FtpServer Class | SocketTools Namespace

FtpServer.OnTimeout Event

Occurs when the client has exceeded the maximum allowed idle time.

[Visual Basic]

```
Public Event OnTimeout As OnTimeoutEventHandler
```

[C#]

```
public event OnTimeoutEventHandler OnTimeout;
```

Event Data

The event handler receives an argument of type FtpServer.TimeoutEventArgs containing data related to this event. The following **FtpServer.TimeoutEventArgs** properties provide information specific to this event.

Property	Description
ClientId	Gets a value that uniquely identifies the client session.
Elapsed	Gets the amount of time that the client was idle.

Remarks

The **OnTimeout** event occurs after the client has has exceeded the maximum allowed idle time, and immediately before the client is disconnected from the server. This event will never occur during a file transfer or directory listing.

To change the default idle timeout period for all clients, set the **IdleTime** property prior to starting the server. To set the idle timeout period for a specific client, set the **ClientIdle** property in an **OnConnect** or **OnLogin** event handler.

This event handler will be invoked in the context of the worker thread that is managing the client session, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.TimeoutEventArgs Class

Provides data for the OnTimeout event.

For a list of all members of this type, see FtpServer.TimeoutEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FtpServer.TimeoutEventArgs

[Visual Basic]

```
Public Class FtpServer.TimeoutEventArgs
    Inherits EventArgs
```

[C#]

```
public class FtpServer.TimeoutEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

FtpServer.TimeoutEventArgs Members | SocketTools Namespace | OnTimeout Event (SocketTools.FtpServer)

FtpServer.TimeoutEventArgs Constructor

Initializes a new instance of the FtpServer.TimeoutEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FtpServer.TimeoutEventArgs();
```

See Also

FtpServer.TimeoutEventArgs Class | SocketTools Namespace

FtpServer.TimeoutEventArgs Members

FtpServer.TimeoutEventArgs overview

Public Instance Constructors

FtpServer.TimeoutEventArgs Constructor	Initializes a new instance of the FtpServer.TimeoutEventArgs class.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
Elapsed	Gets the amount of time that the client was idle.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpServer.TimeoutEventArgs Class | SocketTools Namespace | OnTimeout Event (SocketTools.FtpServer)

FtpServer.TimeoutEventArgs Properties

The properties of the **FtpServer.TimeoutEventArgs** class are listed below. For a complete list of **FtpServer.TimeoutEventArgs** class members, see the FtpServer.TimeoutEventArgs Members topic.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
Elapsed	Gets the amount of time that the client was idle.

See Also

FtpServer.TimeoutEventArgs Class | SocketTools Namespace | OnTimeout Event (SocketTools.FtpServer)

FtpServer.TimeoutEventArgs.ClientId Property

Gets a value that uniquely identifies the client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which identifies the client session.

Remarks

The **ClientId** property returns a unique integer value that identifies the client session that generated the event.

See Also

FtpServer.TimeoutEventArgs Class | SocketTools Namespace

FtpServer.OnUpload Event

Occurs when the client has successfully uploaded a file to the server.

[Visual Basic]

```
Public Event OnUpload As OnUploadEventHandler
```

[C#]

```
public event OnUploadEventHandler OnUpload;
```

Event Data

The event handler receives an argument of type FtpServer.UploadEventArgs containing data related to this event. The following **FtpServer.UploadEventArgs** properties provide information specific to this event.

Property	Description
ClientId	Gets a value that uniquely identifies the client session.
FileName	Gets a value that specifies the file being uploaded.
FileSize	Gets a value that specifies the size of the file.

Remarks

The **OnUpload** event occurs after the client has successfully uploaded a file to the server using the APPE, STOR or STOU command. If the file transfer fails or is aborted, this event will not occur.

This event handler will be invoked in the context of the worker thread that is managing the client session, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

FtpServer Class | SocketTools Namespace

FtpServer.UploadEventArgs Class

Provides data for the OnUpload event.

For a list of all members of this type, see FtpServer.UploadEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.FtpServer.UploadEventArgs

[Visual Basic]
```
Public Class FtpServer.UploadEventArgs
    Inherits EventArgs
```

[C#]
```
public class FtpServer.UploadEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

FtpServer.UploadEventArgs Members | SocketTools Namespace

FtpServer.UploadEventArgs Constructor

Initializes a new instance of the FtpServer.UploadEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public FtpServer.UploadEventArgs();
```

See Also

FtpServer.UploadEventArgs Class | SocketTools Namespace

FtpServer.UploadEventArgs Members

FtpServer.UploadEventArgs overview

Public Instance Constructors

FtpServer.UploadEventArgs Constructor	Initializes a new instance of the FtpServer.UploadEventArgs class.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
FileName	Gets a value that specifies the file being uploaded.
FileSize	Gets a value that specifies the size of the file.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

FtpServer.UploadEventArgs Class | SocketTools Namespace

FtpServer.UploadEventArgs Properties

The properties of the **FtpServer.UploadEventArgs** class are listed below. For a complete list of **FtpServer.UploadEventArgs** class members, see the FtpServer.UploadEventArgs Members topic.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
FileName	Gets a value that specifies the file being uploaded.
FileSize	Gets a value that specifies the size of the file.

See Also

FtpServer.UploadEventArgs Class | SocketTools Namespace

FtpServer.UploadEventArgs.ClientId Property

Gets a value that uniquely identifies the client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which identifies the client session.

Remarks

The **ClientId** property returns a unique integer value that identifies the client session that generated the event.

See Also

FtpServer.UploadEventArgs Class | SocketTools Namespace

FtpServer.UploadEventArgs.FileName Property

Gets a value that specifies the file being uploaded.

[Visual Basic]

```
Public ReadOnly Property FileName As String
```

[C#]

```
public string FileName {get;}
```

Property Value

A string that specifies the full path to a file on the local system.

See Also

FtpServer.UploadEventArgs Class | SocketTools Namespace

FtpServer.UploadEventArgs.FileSize Property

Gets a value that specifies the size of the file.

[Visual Basic]

```
Public ReadOnly Property FileSize As Long
```

[C#]

```
public long FileSize {get;}
```

Property Value

A long integer value that specifies the size of the file in bytes.

See Also

FtpServer.UploadEventArgs Class | SocketTools Namespace

FtpServer.OnAuthenticateEventHandler Delegate

Represents the method that will handle the OnAuthenticate event.

[Visual Basic]

```
Public Delegate Sub FtpServer.OnAuthenticateEventHandler( _
   ByVal sender As Object, _
   ByVal e As AuthenticateEventArgs _
)
```

[C#]

```
public delegate void FtpServer.OnAuthenticateEventHandler(
      object sender,
      AuthenticateEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An AuthenticateEventArgs that contains the event data.

Remarks

When you create an **OnAuthenticateEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnAuthenticateEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

SocketTools Namespace

FtpServer.OnCommandEventHandler Delegate

Represents the method that will handle the OnCommand event.

[Visual Basic]

```
Public Delegate Sub FtpServer.OnCommandEventHandler( _
   ByVal sender As Object, _
   ByVal e As CommandEventArgs _
)
```

[C#]

```
public delegate void FtpServer.OnCommandEventHandler(
      object sender,
      CommandEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An CommandEventArgs that contains the event data.

Remarks

When you create an **OnCommandEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnCommandEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

SocketTools Namespace

FtpServer.OnConnectEventHandler Delegate

Represents the method that will handle the OnConnect event.

[Visual Basic]

```
Public Delegate Sub FtpServer.OnConnectEventHandler( _
   ByVal sender As Object, _
   ByVal e As ConnectEventArgs _
)
```

[C#]

```
public delegate void FtpServer.OnConnectEventHandler(
      object sender,
      ConnectEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ConnectEventArgs that contains the event data.

Remarks

When you create an **OnConnectEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnConnectEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

SocketTools Namespace

FtpServer.OnDisconnectEventHandler Delegate

Represents the method that will handle the OnDisconnect event.

[Visual Basic]

```
Public Delegate Sub FtpServer.OnDisconnectEventHandler( _
   ByVal sender As Object, _
   ByVal e As DisconnectEventArgs _
)
```

[C#]

```
public delegate void FtpServer.OnDisconnectEventHandler(
      object sender,
      DisconnectEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An DisconnectEventArgs that contains the event data.

Remarks

When you create an **OnDisconnectEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnDisconnectEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

SocketTools Namespace

FtpServer.OnDownloadEventHandler Delegate

Represents the method that will handle the OnDownload event.

[Visual Basic]

```
Public Delegate Sub FtpServer.OnDownloadEventHandler( _
   ByVal sender As Object, _
   ByVal e As DownloadEventArgs _
)
```

[C#]

```
public delegate void FtpServer.OnDownloadEventHandler(
      object sender,
      DownloadEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An DownloadEventArgs that contains the event data.

Remarks

When you create an **OnDownloadEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnDownloadEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

SocketTools Namespace

FtpServer.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub FtpServer.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void FtpServer.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ErrorEventArgs that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

SocketTools Namespace

FtpServer.OnExecuteEventHandler Delegate

Represents the method that will handle the OnExecute event.

[Visual Basic]

```
Public Delegate Sub FtpServer.OnExecuteEventHandler( _
   ByVal sender As Object, _
   ByVal e As ExecuteEventArgs _
)
```

[C#]

```
public delegate void FtpServer.OnExecuteEventHandler(
      object sender,
      ExecuteEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ExecuteEventArgs that contains the event data.

Remarks

When you create an **OnExecuteEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnExecuteEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

SocketTools Namespace

FtpServer.OnLoginEventHandler Delegate

Represents the method that will handle the OnLogin event.

[Visual Basic]
```
Public Delegate Sub FtpServer.OnLoginEventHandler( _
   ByVal sender As Object, _
   ByVal e As LoginEventArgs _
)
```

[C#]
```
public delegate void FtpServer.OnLoginEventHandler(
      object sender,
      LoginEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An LoginEventArgs that contains the event data.

Remarks

When you create an **OnLoginEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnLoginEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

SocketTools Namespace

FtpServer.OnLogoutEventHandler Delegate

Represents the method that will handle the OnLogout event.

[Visual Basic]

```
Public Delegate Sub FtpServer.OnLogoutEventHandler( _
   ByVal sender As Object, _
   ByVal e As LogoutEventArgs _
)
```

[C#]

```
public delegate void FtpServer.OnLogoutEventHandler(
      object sender,
      LogoutEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An LogoutEventArgs that contains the event data.

Remarks

When you create an **OnLogoutEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnLogoutEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

SocketTools Namespace

FtpServer.OnResultEventHandler Delegate

Represents the method that will handle the OnResult event.

[Visual Basic]

```
Public Delegate Sub FtpServer.OnResultEventHandler( _
   ByVal sender As Object, _
   ByVal e As ResultEventArgs _
)
```

[C#]

```
public delegate void FtpServer.OnResultEventHandler(
      object sender,
      ResultEventArgs e
   );
```

Parameters

sender
: The source of the event.

e
: An ResultEventArgs that contains the event data.

Remarks

When you create an **OnResultEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnResultEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

SocketTools Namespace

FtpServer.OnTimeoutEventHandler Delegate

Represents the method that will handle the OnTimeout event.

[Visual Basic]

```
Public Delegate Sub FtpServer.OnTimeoutEventHandler( _
   ByVal sender As Object, _
   ByVal e As TimeoutEventArgs _
)
```

[C#]

```
public delegate void FtpServer.OnTimeoutEventHandler(
      object sender,
      TimeoutEventArgs e
   );
```

Parameters

sender
The source of the event.

e
An TimeoutEventArgs that contains the event data.

Remarks

When you create an **OnTimeoutEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnTimeoutEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

SocketTools Namespace

FtpServer.OnUploadEventHandler Delegate

Represents the method that will handle the OnUpload event.

[Visual Basic]

```
Public Delegate Sub FtpServer.OnUploadEventHandler( _
   ByVal sender As Object, _
   ByVal e As UploadEventArgs _
)
```

[C#]

```
public delegate void FtpServer.OnUploadEventHandler(
      object sender,
      UploadEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An UploadEventArgs that contains the event data.

Remarks

When you create an **OnUploadEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnUploadEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

SocketTools Namespace

FtpServer.ErrorCode Enumeration

Specifies the error codes returned by the FtpServer class.

[Visual Basic]
```
Public Enum FtpServer.ErrorCode
```

[C#]
```
public enum FtpServer.ErrorCode
```

Remarks

The FtpServer class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.
errorProxyAuthenticationRequired	Proxy authentication required.

errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.
errorUnsupportedMediaType	Request specified an unsupported media type.

errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.
errorNoMoreMailboxes	No more mailboxes exist on this server.

errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.
errorInvalidThreadHandle	The specified thread handle is no longer valid.

errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been defined.

errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or unrecognized.

errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

FtpServer.FormatType Enumeration

Specifies the logfile formats that the FtpServer class supports.

[Visual Basic]

```
Public Enum FtpServer.FormatType
```

[C#]

```
public enum FtpServer.FormatType
```

Members

Member Name	Description
formatNone	This value specifies that the server should not create or update a log file. This is the default property value.
formatCommon	This value specifies that the log file should use the common log format that records a subset of information in a fixed format. This log format usually only provides information about file transfers.
formatExtended	This value specifies that the log file should use the standard W3C extended log file format. This is an extensible format that can provide additional information about the client session.

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

SocketTools Namespace

FtpServer.SecurityProtocols Enumeration

Specifies the security protocols that the FtpServer class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum FtpServer.SecurityProtocols
```

[C#]

```
[Flags]
public enum FtpServer.SecurityProtocols
```

Remarks

The FtpServer class uses the **SecurityProtocols** enumeration to specify one or more security protocols to be used when establishing a connection with a remote host. Multiple protocols may be specified if necessary and the actual protocol used will be negotiated with the remote host. It is recommended that most applications use **protocolDefault** when starting a secure server.

Members

Member Name	Description	Value
protocolNone	No security protocol will be used, a secure connection will not be established.	0
protocolSSL2	The SSL 2.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections.	1
protocolSSL3	The SSL 3.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections. In most cases, this protocol is only selected if TLS is not supported by the server.	2
protocolTLS10	The TLS 1.0 protocol has been selected. This version of the protocol is commonly used by older servers and is the only version of TLS supported on Windows platforms prior to Windows 7 SP1 and Windows Server 2008 R2.	4
protocolTLS11	The TLS 1.1 protocol has been selected. This version of TLS is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of the	8

	operating system.	
protocolTLS12	The TLS 1.2 protocol has been selected. This is the default version of the protocol and is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of Windows. It is recommended that you use this version of TLS.	16
protocolSSL	Any version of the Secure Sockets Layer (SSL) protocol should be used. The actual protocol version used will be negotiated with the remote host.	3
protocolTLS	Any version of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with with preference for TLS 1.2.	28
protocolTLS1	Version 1.0, 1.1 or 1.2 of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with preference for TLS 1.2.	28
protocolDefault	The default selection of security protocols will be used when establishing a connection. The TLS 1.2, 1.1 and 1.0 protocols will be negotiated with the host, in that order of preference. This option will always request the latest version of the preferred security protocols and is the recommended value.	16
protocolUnknown	An unknown or unsupported security protocol has been specified. This value indicates an error condition.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

SocketTools Namespace

FtpServer.ServerOptions Enumeration

Specifies the options that the FtpServer class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum FtpServer.ServerOptions
```

[C#]

```
[Flags]
public enum FtpServer.ServerOptions
```

Remarks

The FtpServer class uses the **ServerOptions** enumeration to specify one or more options to be used when establishing a connection with a remote host. Multiple options may be specified if necessary.

Members

Member Name	Description	Value
optionNone	No option specified.	0
optionMultiUser	This option specifies the server should be started in multi-user mode, where users are provided with their own home directories based on their username. If this option is not specified, then all users will share the server root directory by default. This option does not affect the maximum number of simultaneous client connections to the server. To isolate users to their own individual home directory, combine this option with the **optionRestricted** option	1
optionRestricted	This option specifies the server should be initialized in a restricted mode that isolates the server and limits the ability for clients to access files on the host system. All file transfers are limited to the user's home directory. This option also disables certain site-specific commands. This is a recommended option for general purpose applications designed to accept connections from clients over the Internet. This option is only meaningful if the **optionMultiUser** option has also been specified. All clients are restricted to the server root directory and its subdirectories, regardless of whether this option is specified or not	2

optionLocalUser	This option specifies the server should perform user authentication using the Windows local account database. This option is useful if the server should accept local usernames, or if the application does not wish to implement an event handler for user authentication. If this option is not specified, the application is responsible for authenticating all users.	4
optionAnonymous	This option specifies the server should accept anonymous client connections. This is typically used to provide public access to files without requiring the client to have valid credentials on the server. Anonymous clients are automatically authenticated by the server, but are restricted to a public directory and subdirectories. If this option is enabled, it is recommended that you also specify the **optionReadOnly** option to prevent anonymous clients from uploading files to the server.	8
optionReadOnly	This option specifies the server should only allow read-only access to files by default. If this option is enabled, it will change the default permissions granted to authenticated users. Anonymous clients will not be able to upload, rename or delete files and cannot create subdirectories. It is recommended that this option be enabled if the server is publicly accessible over the Internet and the **optionAnonymous** option has been specified.	16
optionLocalTime	This option specifies the server should return file and directory times adjusted for the local timezone. By default, the server will return all file times as UTC values. This option affects the time information sent to a client when a list of files is requested, as well as when status information is requested for a specific file. This option will not affect the MDTM and MFMT commands which always use file times as UTC values.	32
optionLockFiles	This option specifies that files should be exclusively locked when a client	64

	attempts to upload or download a file. If another client attempts to access the same file, the operation will fail. By default, the server will permit multiple clients to access the same file, although it will still write-lock files that are in the process of being uploaded.	
optionHiddenFiles	This option specifies that when a client requests a list of files in a directory, the server should include any hidden and system files files or subdirectories. By default, the server will not include hidden or system files, although they are still accessible to the client if it knows the name of the file. File names that begin with a period are also considered to be hidden files and will not normally be included in file listings.	128
optionUnixMode	This option specifies the server should impersonate a UNIX-based operating system. The server will identify itself as running on a UNIX system and directory listings will be in a format commonly used by UNIX. If this option is not specified, the server will identify itself as running on Windows NT and directory listings will be in the same format used by the Microsoft IIS FTP server. Note that this option does not affect the path delimiter used with file and directory names.	256
optionExternal	This option specifies the server is listening for client connections from behind a router that uses Network Address Translation (NAT). If enabled, the server will report its external IP address rather than the address assigned to it on the local network. For the server to accept connections from behind a NAT router, the router must be configured to direct inbound traffic to the specified port number on the host system.	512
optionSecure	This option specifies that secure connections using SSL and/or TLS should be enabled. If neither the **optionExplicitSSL** or **optionImplicitSSL** options are specified, the server automatically determines the appropriate type based	4096

	on the port number. If the local port number is 990, then implicit SSL will be used, otherwise explicit SSL will be used. This option requires that a valid SSL certificate be installed on the local host.	
optionExplicitSSL	This option specifies the server will accept the AUTH TLS command and negotiate a secure connection with the client after that command is issued. This option implies the **optionSecure** option and requires that a valid SSL certificate be installed on the local host.	8192
optionImplicitSSL	This option specifies the server should negotiate a secure connection with the client immediately after it connects to the server. It is recommended that you only use this option if the server is listening for connections on port 990, which is the standard port for FTP servers using implicit SSL. This option implies the **optionSecure** option and requires that a valid SSL certificate be installed on the local host.	16384
optionSecureFallback	This option specifies the server should permit the use of less secure cipher suites for compatibility with legacy clients. If this option is specified, the server will permit connections using TLS 1.0 and cipher suites that use RC4, MD5 and SHA1.	32768

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

SocketTools Namespace

FtpServer.ServerPriority Enumeration

Specifies the priorities that the FtpServer class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum FtpServer.ServerPriority
```

[C#]
```
[Flags]
public enum FtpServer.ServerPriority
```

Members

Member Name	Description	Value
priorityBackground	This priority significantly reduces the memory, processor and network resource utilization for the server. It is typically used with lightweight services running in the background that are designed for few client connections. Each client thread will be assigned a lower scheduling priority and will be frequently forced to yield execution to other threads.	0
priorityLow	This priority lowers the overall resource utilization for the client session and meters the processor utilization for the client session. Each client thread will be assigned a lower scheduling priority and will occasionally be forced to yield execution to other threads.	1
priorityNormal	The default priority which balances resource and processor utilization. It is recommended that most applications use this priority.	2
priorityHigh	This priority increases the overall resource utilization for each client session and their threads will be given higher scheduling priority. It is not recommended that this priority be used on a system with a single processor.	3
priorityCritical	This priority can significantly increase processor, memory and network utilization. Each client thread will be given higher scheduling priority and will be more responsive to network events. It is not recommended that this priority be used on a system with a single	4

	processor.	
priorityInvalid	An invalid transfer priority which indicates an error condition.	-1
priorityDefault	The default server priority. This is the same as specifying **priorityNormal**.	2
priorityLowest	The lowest valid server priority. This is the same as specifying **priorityBackground**.	0
priorityHighest	The highest valid server priority. This is the same as specifying **priorityCritical**.	4

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

SocketTools Namespace

FtpServer.TraceOptions Enumeration

Specifies the logging options that the FtpServer class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum FtpServer.TraceOptions
```

```
[C#]
[Flags]
public enum FtpServer.TraceOptions
```

Remarks

The FtpServer class uses the **TraceOptions** enumeration to specify what kind of debugging information is written to the trace logfile. These options are only meaningful when trace logging is enabled by setting the **Trace** property to **true**.

Members

Member Name	Description	Value
traceDefault	The default trace logging option. This is the same as specifying the **traceInfo** option.	0
traceInfo	All network function calls are written to the trace file. This is the default value.	0
traceError	Only those network function calls which fail are recorded in the trace file.	1
traceWarning	Only those network function calls which fail, or return values which indicate a warning, are recorded in the trace file.	2
traceHexDump	All network function calls are written to the trace file, plus all the data that is sent or received is displayed, in both ASCII and hexadecimal format.	4
traceProcess	All function calls in the current process are logged, rather than only those functions in the current thread. This option is useful for multithreaded applications that are using worker threads.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.FtpServer (in SocketTools.FtpServer.dll)

See Also

SocketTools Namespace

HttpClient Class

Implements the Hypertext Transfer Protocol.

For a list of all members of this type, see HttpClient Members.

System.Object
  SocketTools.HttpClient

[Visual Basic]
```
Public Class HttpClient
    Implements IDisposable
```

[C#]
```
public class HttpClient : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Example

The Hypertext Transfer Protocol (HTTP) is a lightweight, stateless application protocol that is used to access resources on web servers, as well as send data to those servers for processing. The HttpClient class provides direct, low-level access to the server and the commands that are used to retrieve resources (i.e.: documents, images, etc.). The class also provides a simple interface for downloading resources to the local host, similar to how the FtpClient class can be used to download files.

In a typical session, the class is used to establish a connection, send a request (to download a resource, post data for processing, etc.), read the data returned by the server and then disconnect. It is the responsibility of the client to process the data returned by the server, depending on the type of resource that was requested. For example, if an HTML document was requested, the client may parse the contents of the file, looking for specific information.

This class supports secure connections using the standard TLS protocols.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

HttpClient Members | SocketTools Namespace

HttpClient Members

HttpClient overview

Public Static (Shared) Fields

httpPortDefault	A constant value which specifies the default port number.
httpPortSecure	A constant value which specifies the default port number for a secure connection.
httpTimeout	A constant value which specifies the default timeout period.

Public Static (Shared) Methods

ErrorText	Returns the description of an error code.

Public Instance Constructors

HttpClient Constructor	Initializes a new instance of the HttpClient class.

Public Instance Fields

CookieName	Gets a string which specifies the name of a cookie returned by the server.
CookieValue	Gets a string which specifies the value of a cookie returned by the server.

Public Instance Properties

Authentication	Gets and sets the method used to authenticate the client session.
AutoRedirect	Gets and sets a value that specifies if redirected resources are handled automatically.
AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
BearerToken	Gets and sets the bearer token used with OAuth 2.0 authentication.
Blocking	Gets and sets a value which indicates if the client is in blocking mode.
CertificateExpires	Get a value that specifies the date that the security certificate expires.
CertificateIssued	Get a value that specifies the date that the security certificate was issued.
CertificateIssuer	Get a value that provides information about the organization that issued the certificate.
CertificateName	Gets and sets a value that specifies the name of the client certificate.

CertificatePassword	Gets and sets the password associated with the client certificate.
CertificateStatus	Gets a value which indicates the status of the security certificate returned by the remote host.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateSubject	Gets a value that provides information about the organization that the server certificate was issued to.
CertificateUser	Gets and sets the user that owns the client certificate.
CipherStrength	Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.
Compression	Gets and sets a value that specifies if data compression should be enabled.
CookieCount	Gets the number of cookies set by the server in response to a request for a resource.
Encoding	Gets and sets the content encoding type.
FormAction	Gets and sets the path to the script that will accept the form data on the server.
FormMethod	Gets and sets the method used to submit the form data.
FormType	Gets and sets the type of encoding used to submit the form data.
Handle	Gets a value that specifies the client handle allocated for the current session.
HashStrength	Gets a value which specifies the length of the message digest that was selected for a secure connection.
HeaderField	Gets and sets the name of the current header field.
HeaderValue	Gets the value of a response header field or sets the value of a request header field.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsConnected	Gets a value which indicates if a connection to the remote host has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.

IsReadable	Gets a value which indicates if there is data available to be read from the socket connection to the server.
IsWritable	Gets a value which indicates if data can be written to the client without blocking.
KeepAlive	Gets and sets a Boolean value that specifies if the connection to the server is persistent.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets the local Internet address that the client is bound to.
Localize	Gets and sets a value which specifies if date values are localized to the current timezone.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
Options	Gets and sets a value which specifies one or more client options.
Password	Gets and sets the password used to authenticate the client session.
Priority	Gets and sets a value which specifies the priority of data transfers.
ProtocolVersion	Gets and sets a value which specifies the default protocol version.
ProxyHost	Gets and sets the hostname or IP address of a proxy server.
ProxyPassword	Gets and sets the password used to authenticate the connection to a proxy server.
ProxyPort	Gets and sets a value that specifies the proxy server port number.
ProxyType	Gets and sets the type of proxy server the client will use to establish a connection.
ProxyUser	Gets and sets the username used to authenticate the connection to a proxy server.
RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.
Resource	Gets and sets a value which specifies a resource on the server.

ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.
Secure	Gets and sets a value which specifies if a secure connection is established.
SecureCipher	Gets a value that specifies the encryption algorithm used for a secure connection.
SecureHash	Gets a value that specifies the message digest algorithm used for a secure connection.
SecureKeyExchange	Gets a value that specifies the key exchange algorithm used for a secure connection.
SecureProtocol	Gets and sets a value which specifies the protocol used for a secure connection.
Status	Gets a value which specifies the current status of the client.
TaskCount	Get the number of active background file transfers.
TaskId	Get the task identifier for the last background file transfer.
TaskList	Get an array of active background task identifiers.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
TimeZone	Gets and sets the current timezone offset in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
TransferBytes	Gets a value which specifies the number of bytes transferred to or from the server.
TransferRate	Gets a value which specifies the data transfer rate in bytes per second.
TransferTime	Gets a value which specifies the number of seconds elapsed during a data transfer.
URL	Gets and sets the current URL used to access a file

	on the server.
UserAgent	Gets and sets the current user agent value which identifies the application.
UserName	Gets and sets the username used to authenticate the client session.
Version	Gets a value which returns the current version of the HttpClient class library.

Public Instance Methods

AddField	Overloaded. Add the form field and its value to the current form.
AddFile	Append the contents of the file to the current form.
AsyncGetFile	Overloaded. Download a file from the server to the local system in the background.
AsyncPutFile	Overloaded. Upload a file from the local system to the server in the background.
AttachThread	Attach an instance of the class to the current thread
Authenticate	Overloaded. Authenticate the client session with a username and password.
Cancel	Cancel the current blocking client operation.
ClearForm	Remove all defined fields from the current form.
ClearHeaders	Clears the current request and response headers.
CloseFile	Close the file that was opened on the server.
Command	Overloaded. Send a custom command to the web server.
Connect	Overloaded. Establish a connection with a remote host.
CreateFile	Overloaded. Create a new file or overwrite an existing file on the web server.
CreateForm	Overloaded. Create a new virtual form.
DeleteField	Delete a form field and its value from the current form.
DeleteFile	Remove a file on the web server.
Disconnect	Terminate the connection with a remote host.
Dispose	Overloaded. Releases all resources used by HttpClient.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetCookie	Overloaded. Return information about the

	specified cookie.
GetData	Overloaded. Transfer data from the web server and store it in a local buffer.
GetFile	Overloaded. Transfer data from the web server and store it in a file on the local system.
GetFileSize	Overloaded. Return the size of the specified file on the web server.
GetFileTime	Overloaded. Return the modification date and time for specified file on the web server.
GetFirstHeader	Return the first response header field name and value.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetHeader	Return the value of the specified response header field.
GetNextHeader	Return the next response header field name and value.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the HttpClient class.
OpenFile	Open a file on the web server for reading.
PatchData	Overloaded. Submits patch data to the server and returns the result in a string.
PostData	Overloaded. Submits the contents of the specified buffer to a resource on the server.
PostFile	Overloaded. Upload the contents of a file to a resource on the server.
PostJson	Overloaded. Submits JSON formatted data to the server and returns the result in a string.
PostXml	Overloaded. Submits XML formatted data to the server and returns the result in a string.
PutData	Overloaded. Transfer data from a local buffer to the server.
PutFile	Overloaded. Transfer a file from the local system to the web server.
Read	Overloaded. Read data from the server and store it in a byte array.
Reset	Reset the internal state of the object, resetting all properties to their default values.
SetCookie	Send the specified cookie to the server when a resource is requested.

SetHeader	Set the value of a request header field.
SubmitForm	Overloaded. Submits the current form data to the server for processing.
TaskAbort	Overloaded. Abort the specified asynchronous task.
TaskDone	Overloaded. Determine if an asynchronous task has completed.
TaskResume	Overloaded. Resume execution of an asynchronous task.
TaskSuspend	Overloaded. Suspend execution of an asynchronous task.
TaskWait	Overloaded. Wait for an asynchronous task to complete.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
Write	Overloaded. Write one or more bytes of data to the server.

Public Instance Events

OnCancel	Occurs when a blocking client operation is canceled.
OnCommand	Occurs when the client sends a command to the remote host and receives a reply indicating the result of that command.
OnConnect	Occurs when a connection is established with the remote host.
OnDisconnect	Occurs when the remote host disconnects from the local system.
OnError	Occurs when an client operation fails.
OnProgress	Occurs as a data stream is being read or written to the client.
OnRead	Occurs when data is available to be read from the client.
OnRedirect	Occurs when the server indicates a resource has been moved.
OnTaskBegin	Occurs when an asynchronous task begins execution.
OnTaskEnd	Occurs when an asynchronous task completes.
OnTaskRun	Occurs while a background task is active.
OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.

OnWrite	Occurs when data can be written to the client.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the HttpClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpClient Class | SocketTools Namespace

HttpClient Constructor

Initializes a new instance of the HttpClient class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public HttpClient();
```

See Also

HttpClient Class | SocketTools Namespace

HttpClient Fields

The fields of the **HttpClient** class are listed below. For a complete list of **HttpClient** class members, see the HttpClient Members topic.

Public Static (Shared) Fields

httpPortDefault	A constant value which specifies the default port number.
httpPortSecure	A constant value which specifies the default port number for a secure connection.
httpTimeout	A constant value which specifies the default timeout period.

Public Instance Fields

CookieName	Gets a string which specifies the name of a cookie returned by the server.
CookieValue	Gets a string which specifies the value of a cookie returned by the server.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.CookieName Field

Gets a string which specifies the name of a cookie returned by the server.

[Visual Basic]

```
Public ReadOnly CookieName As CookieNameArray
```

[C#]

```
public readonly CookieNameArray CookieName;
```

Remarks

The **CookieName** array returns a string which identifies the cookie specified by the ***index*** argument. The array is zero based, which means the name of the first available cookie is read by using an index value of zero. The **CookieCount** property indicates the total number of cookies that have been returned by the server.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.CookieValue Field

Gets a string which specifies the value of a cookie returned by the server.

[Visual Basic]

```
Public ReadOnly CookieValue As CookieValueArray
```

[C#]

```
public readonly CookieValueArray CookieValue;
```

Remarks

The **CookieValue** array returns a string which contains the value for the cookie specified by the ***index*** argument. The array is zero based, which means the value of the first available cookie is read by using an index value of zero. The **CookieCount** property indicates the total number of cookies that have been returned by the server.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.HttpCookie Structure

This structure is used by the GetCookie method to return information about a cookie set by the server.

For a list of all members of this type, see HttpClient.HttpCookie Members.

System.Object
  System.ValueType
    SocketTools.HttpClient.HttpCookie

[Visual Basic]
```
Public Structure HttpClient.HttpCookie
```

[C#]
```
public struct HttpClient.HttpCookie
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

HttpClient.HttpCookie Members | SocketTools Namespace

HttpClient.HttpCookie Members

HttpClient.HttpCookie overview

Public Instance Fields

Domain	Gets a value which specifies the cookie domain.
Expires	Gets a value which specifies the date and time the cookie expires.
Flags	Gets a value which specifies one or more cookie flags.
Name	Gets a value which specifies the name of the cookie.
Path	Gets a value which specifies the cookie path.
Value	Gets a value which specifies the contents of the cookie.

Public Instance Methods

Equals (inherited from ValueType)	Indicates whether this instance and a specified object are equal.
GetHashCode (inherited from ValueType)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from ValueType)	Returns the fully qualified type name of this instance.

See Also

HttpClient.HttpCookie Class | SocketTools Namespace

HttpClient.HttpCookie Fields

The fields of the **HttpClient.HttpCookie** structure are listed below. For a complete list of **HttpClient.HttpCookie** structure members, see the HttpClient.HttpCookie Members topic.

Public Instance Fields

Domain	Gets a value which specifies the cookie domain.
Expires	Gets a value which specifies the date and time the cookie expires.
Flags	Gets a value which specifies one or more cookie flags.
Name	Gets a value which specifies the name of the cookie.
Path	Gets a value which specifies the cookie path.
Value	Gets a value which specifies the contents of the cookie.

See Also

HttpClient.HttpCookie Class | SocketTools Namespace

HttpClient.HttpCookie.Domain Field

Gets a value which specifies the cookie domain.

[Visual Basic]

```
Public Domain As String
```

[C#]

```
public string Domain;
```

Remarks

The cookie domain specifies the domain for which the cookie should be used. Matches are made by comparing the name of the remote host against the domain name specified in the cookie. If the domain is example.com, then any server in the example.com domain would match; for example, both shipping.example.com and orders.example.com would match the domain value. However, if the cookie domain was orders.example.com, then the cookie would only be sent if the resource was requested from orders.example.com, not if the resource was located on shipping.example.com or www.example.com.

See Also

HttpClient.HttpCookie Class | SocketTools Namespace

HttpClient.HttpCookie.Expires Field

Gets a value which specifies the date and time the cookie expires.

[Visual Basic]

```
Public Expires As Date
```

[C#]

```
public DateTime Expires;
```

Remarks

If the cookie expiration value is later than the current date and time, the cookie should not be provided to the server when a resource is requested. This is only valid for persistent cookies, since session cookies are automatically deleted when the client application terminates. The time is always expressed as Coordinated Universal Time.

See Also

HttpClient.HttpCookie Class | SocketTools Namespace

HttpClient.HttpCookie.Flags Field

Gets a value which specifies one or more cookie flags.

[Visual Basic]

```
Public Flags As CookieFlags
```

[C#]

```
public CookieFlags Flags;
```

Remarks

The cookie flags value provides additional information about the cookie. In some cases, a cookie should only be submitted to the server if the resource is requested using a secure connection. In this case, the bit flag **cookieSecure** will be set

See Also

HttpClient.HttpCookie Class | SocketTools Namespace

HttpClient.HttpCookie.Name Field

Gets a value which specifies the name of the cookie.

[Visual Basic]

```
Public Name As String
```

[C#]

```
public string Name;
```

See Also

HttpClient.HttpCookie Class | SocketTools Namespace

HttpClient.HttpCookie.Path Field

Gets a value which specifies the cookie path.

[Visual Basic]

```
Public Path As String
```

[C#]

```
public string Path;
```

Remarks

The cookie path specifies a path for the resources where the cookie should be used. For example, a path of "/" indicates that the cookie should be provided for all resources requested from the server. A path of "/data" would mean that the cookie should be included if the resource is found in the /data folder or a sub-folder, such as /data/projections.asp. However, the cookie would not be provided if the resource /info/status.asp was requested, since it is not in the /data path.

See Also

HttpClient.HttpCookie Class | SocketTools Namespace

HttpClient.HttpCookie.Value Field

Gets a value which specifies the contents of the cookie.

[Visual Basic]

```
Public Value As String
```

[C#]

```
public string Value;
```

See Also

HttpClient.HttpCookie Class | SocketTools Namespace

HttpClient Properties

The properties of the **HttpClient** class are listed below. For a complete list of **HttpClient** class members, see the HttpClient Members topic.

Public Instance Properties

Authentication	Gets and sets the method used to authenticate the client session.
AutoRedirect	Gets and sets a value that specifies if redirected resources are handled automatically.
AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
BearerToken	Gets and sets the bearer token used with OAuth 2.0 authentication.
Blocking	Gets and sets a value which indicates if the client is in blocking mode.
CertificateExpires	Get a value that specifies the date that the security certificate expires.
CertificateIssued	Get a value that specifies the date that the security certificate was issued.
CertificateIssuer	Get a value that provides information about the organization that issued the certificate.
CertificateName	Gets and sets a value that specifies the name of the client certificate.
CertificatePassword	Gets and sets the password associated with the client certificate.
CertificateStatus	Gets a value which indicates the status of the security certificate returned by the remote host.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateSubject	Gets a value that provides information about the organization that the server certificate was issued to.
CertificateUser	Gets and sets the user that owns the client certificate.
CipherStrength	Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.
Compression	Gets and sets a value that specifies if data compression should be enabled.
CookieCount	Gets the number of cookies set by the server in response to a request for a resource.
Encoding	Gets and sets the content encoding type.

FormAction	Gets and sets the path to the script that will accept the form data on the server.
FormMethod	Gets and sets the method used to submit the form data.
FormType	Gets and sets the type of encoding used to submit the form data.
Handle	Gets a value that specifies the client handle allocated for the current session.
HashStrength	Gets a value which specifies the length of the message digest that was selected for a secure connection.
HeaderField	Gets and sets the name of the current header field.
HeaderValue	Gets the value of a response header field or sets the value of a request header field.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsConnected	Gets a value which indicates if a connection to the remote host has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsReadable	Gets a value which indicates if there is data available to be read from the socket connection to the server.
IsWritable	Gets a value which indicates if data can be written to the client without blocking.
KeepAlive	Gets and sets a Boolean value that specifies if the connection to the server is persistent.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets the local Internet address that the client is bound to.
Localize	Gets and sets a value which specifies if date values are localized to the current timezone.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.

Options	Gets and sets a value which specifies one or more client options.
Password	Gets and sets the password used to authenticate the client session.
Priority	Gets and sets a value which specifies the priority of data transfers.
ProtocolVersion	Gets and sets a value which specifies the default protocol version.
ProxyHost	Gets and sets the hostname or IP address of a proxy server.
ProxyPassword	Gets and sets the password used to authenticate the connection to a proxy server.
ProxyPort	Gets and sets a value that specifies the proxy server port number.
ProxyType	Gets and sets the type of proxy server the client will use to establish a connection.
ProxyUser	Gets and sets the username used to authenticate the connection to a proxy server.
RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.
Resource	Gets and sets a value which specifies a resource on the server.
ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.
Secure	Gets and sets a value which specifies if a secure connection is established.
SecureCipher	Gets a value that specifies the encryption algorithm used for a secure connection.
SecureHash	Gets a value that specifies the message digest algorithm used for a secure connection.
SecureKeyExchange	Gets a value that specifies the key exchange algorithm used for a secure connection.
SecureProtocol	Gets and sets a value which specifies the protocol used for a secure connection.
Status	Gets a value which specifies the current status of the client.
TaskCount	Get the number of active background file transfers.
TaskId	Get the task identifier for the last background file

	transfer.
TaskList	Get an array of active background task identifiers.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
TimeZone	Gets and sets the current timezone offset in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
TransferBytes	Gets a value which specifies the number of bytes transferred to or from the server.
TransferRate	Gets a value which specifies the data transfer rate in bytes per second.
TransferTime	Gets a value which specifies the number of seconds elapsed during a data transfer.
URL	Gets and sets the current URL used to access a file on the server.
UserAgent	Gets and sets the current user agent value which identifies the application.
UserName	Gets and sets the username used to authenticate the client session.
Version	Gets a value which returns the current version of the HttpClient class library.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.Authentication Property

Gets and sets the method used to authenticate the client session.

[Visual Basic]

```
Public Property Authentication As HttpAuthentication
```

[C#]

```
public HttpClient.HttpAuthentication Authentication {get; set;}
```

Property Value

A HttpAuthentication enumeration value which specifies the authentication method.

Remarks

By default, no authentication is used when accessing a resource on the web server. Setting the **UserName** or **Password** property will automatically set the authentication type to use **authBasic** unless a different method has already been specified.

Changing the value of the **BearerToken** property will automatically set the current authentication method to use **authBearer**.

You should only use the **authBearer** authentication method if you understand the process of how to request the access token. Obtaining an access token requires registering your application with the web service provider (e.g.: Microsoft or Google), getting a unique client ID associated with your application and then requesting the access token using the appropriate scope for the service. Obtaining the initial token will typically involve interactive confirmation on the part of the user, requiring they grant permission to your application to access the service.

See Also

HttpClient Class | SocketTools Namespace | BearerToken Poperty | Password Property | UserName Property | Connect Method

HttpClient.AutoRedirect Property

Gets and sets a value that specifies if redirected resources are handled automatically.

[Visual Basic]
```
Public Property AutoRedirect As Boolean
```

[C#]
```
public bool AutoRedirect {get; set;}
```

Property Value

A boolean value. A value of **true** specifies that requests for resources that have moved will automatically redirect the client to the new location for that resource. A value of **false** specifies that the client is responsible for requesting the resource from the new location.

Remarks

When the server indicates that a resource has been redirected, the **OnRedirect** event will fire and will provide the new location for the resource as an argument to the event handler. It is permissible for the application to change the value of the **AutoRedirect** property inside the event handler to determine whether or not the class will automatically access the resource from the new location.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.AutoResolve Property

Gets and sets a value that determines if host names and addresses are automatically resolved.

[Visual Basic]
```
Public Property AutoResolve As Boolean
```

[C#]
```
public bool AutoResolve {get; set;}
```

Property Value

Returns **true** if host names are automatically resolved to Internet addresses. The default value is **false**.

Remarks

Setting the **AutoResolve** property determines if the class automatically resolves host names and addresses specified by the **HostName** and **HostAddress** properties. If set to **true**, setting the **HostName** property will cause the class to automatically determine the corresponding IP address and update the **HostAddress** property accordingly. Likewise, setting the **HostAddress** property will cause the class to determine the host name and update the **HostName** property. Setting this property to **false** prevents the class from resolving host names until a connection attempt is made.

It is important to note that setting the **HostName** or **HostAddress** property may cause the current thread to block, sometimes for several seconds, until the name or address is resolved. To prevent this behavior, set this property value to **false**.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.BearerToken Property

Gets and sets the bearer token used with OAuth 2.0 authentication.

[Visual Basic]

```
Public Property BearerToken As String
```

[C#]

```
public string BearerToken {get; set;}
```

Property Value

Returns a string which contains the OAuth 2.0 bearer token. Assigning a value to this property sets the curent authentication type to use OAuth 2.0 and updates the bearer token.

Remarks

Assigning a value to the **BearerToken** property will automatically change the current authentication method to use **HttpAuthentication.authBearer** if necessary.

Obtaining a bearer token requires registering your application with the web service provider (e.g.: Microsoft or Google), getting a unique client ID associated with your application and then requesting the bearer token using the appropriate scope for the service. Obtaining the initial token will typically involve interactive confirmation on the part of the user, requiring they grant permission to your application to access the service.

Your application should not store the bearer token for later use. They have a relatively short lifespan, typically about an hour, and are designed to be used with the current client session. You should specify offline access as part of the OAuth 2.0 scope if necessary and store the refresh token provided by the service. The refresh token has a much longer validity period and can be used to obtain a new access token when needed.

If the current authentication method does not use OAuth 2.0, this property will return an empty string and you should use the **Password** property to obtain the current user password.

See Also

HttpClient Class | SocketTools Namespace | Authentication Property | Password Property | UserName Property

HttpClient.Blocking Property

Gets and sets a value which indicates if the client is in blocking mode.

[Visual Basic]

```
Public Property Blocking As Boolean
```

[C#]

```
public bool Blocking {get; set;}
```

Property Value

Returns **true** if the client is in blocking mode; otherwise it returns **false**. The default value is **true**.

Remarks

Setting the **Blocking** property determines if client operations complete synchronously or asynchronously. If set to **true**, then each client operation (such as sending or receiving data) will return when the operation has completed or timed-out. If set to **false**, client operations will return immediately. If the operation would result in the client blocking (such as attempting to read data when no data has been sent by the remote host), an error is generated.

It is important to note that certain events, such as **OnDisconnect, OnRead** and **OnWrite** are only fired if the client is in non-blocking mode.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.CertificateExpires Property

Get a value that specifies the date that the security certificate expires.

[Visual Basic]

```
Public ReadOnly Property CertificateExpires As String
```

[C#]

```
public string CertificateExpires {get;}
```

Property Value

A string which specifies a date using the local date and time format.

Remarks

The **CertificateExpires** property returns a string that specifies the date and time that the security certificate expires. This property will return an empty string if a secure connection has not been established with the remote host.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.CertificateIssued Property

Get a value that specifies the date that the security certificate was issued.

[Visual Basic]

```
Public ReadOnly Property CertificateIssued As String
```

[C#]

```
public string CertificateIssued {get;}
```

Property Value

A string which specifies a date using the local date and time format.

Remarks

The **CertificateIssued** property returns a string that specifies the date and time that the security certificate was issued. This property will return an empty string if a secure connection has not been established with the remote host.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.CertificateIssuer Property

Get a value that provides information about the organization that issued the certificate.

[Visual Basic]

```
Public ReadOnly Property CertificateIssuer As String
```

[C#]

```
public string CertificateIssuer {get;}
```

Property Value

A string that contains a comma separated list of name value pairs.

Remarks

The **CertificateIssuer** property returns a string that contains information about the organization that issued the server certificate. The string value is a comma separated list of tagged name and value pairs. In the nomenclature of the X.500 standard, each of these pairs are called a relative distinguished name (RDN), and when concatenated together, forms the issuer's distinguished name (DN). For example:

```
C=US, O="RSA Data Security, Inc.", OU=Secure Server Certification Authority
```

To obtain a specific value, such as the name of the issuer or the issuer's country, the application must parse the string returned by this property. Some of the common tokens used in the distinguished name are:

Token	Description
C	The ISO standard two character country code.
S	The name of the state or province.
L	The name of the city or locality.
O	The name of the company or organization.
OU	The name of the department or organizational unit
CN	The common name; with X.509 certificates, this is the domain name of the site the certificate was issued for.

This property will return an empty string if a secure connection has not been established with the remote host.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.CertificateName Property

Gets and sets a value that specifies the name of the client certificate.

[Visual Basic]

```
Public Property CertificateName As String
```

[C#]

```
public string CertificateName {get; set;}
```

Property Value

A string which specifies the certificate name.

Remarks

The **CertificateName** property is used to specify the name of a client certificate to use when establishing a secure connection. It is only required that you set this property value if the server requires a client certificate for authentication. If this property is not set, a client certificate will not be provided to the server. If a certificate name is specified, the certificate must have a private key associated with it, otherwise the connection attempt will fail because the control will be unable to create a security context for the session.

When the certificate store is searched for a matching certificate, it will first search for any certificate with a friendly name that matches the property value. If no valid certificate is found, it will then search for a certificate with a matching common name.

Certificates may be installed and viewed on the local system using the Certificate Manager that is included with the Windows operating system. For more information, refer to the documentation for the Microsoft Management Console.

See Also

HttpClient Class | SocketTools Namespace | CertificateStore Property | Secure Property

HttpClient.CertificateStatus Property

Gets a value which indicates the status of the security certificate returned by the remote host.

[Visual Basic]

```
Public ReadOnly Property CertificateStatus As SecurityCertificate
```

[C#]

```
public HttpClient.SecurityCertificate CertificateStatus {get;}
```

Property Value

A SecurityCertificate enumeration value which specifies the status of the certificate.

Remarks

The **CertificateStatus** property is used to determine the status of the security certificate returned by the remote host when a secure connection has been established. This property value should be checked after the connection to the server has completed, but prior to beginning a transaction.

Note that if the certificate cannot be validated, the secure connection will not be automatically terminated. It is the responsibility of your application to determine the best course of action to take if the certificate is invalid. Even if the security certificate cannot be validated, the data exchanged with the remote host will still be encrypted.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.CertificateStore Property

Gets and sets a value that specifies the name of the local certificate store.

[Visual Basic]

```
Public Property CertificateStore As String
```

[C#]

```
public string CertificateStore {get; set;}
```

Property Value

A string which specifies the certificate store name. The default value is the current user's personal certificate store.

Remarks

The **CertificateStore** property is used to specify the name of the certificate store which contains the security certificate to use when establishing a secure connection. The certificate may either be stored in the registry or in a file. If the certificate is stored in the registry, then this property should be set to one of the following predefined values:

Store Name	Description
CA	Certification authority certificates. These are certificates that are issued by entities which are entrusted to issue certificates to other individuals or organizations. Companies such as VeriSign and Thawte act as certification authorities.
MY	Personal certificates and their associated private keys for the current user. This store typically holds the client certificates used to establish a user's credentials. If a certificate store is not specified, this is the default value that is used.
ROOT	Certificates that have been self-signed by a certificate authority. Root certificates for a number of different certification authorities such as VeriSign and Thawte are installed as part of the operating system and periodically updated by Microsoft.

In most cases the client certificate will be installed in the user's personal certificate store, and therefore it is not necessary to set this property value because that is the default location that will be used to search for the certificate. This property is only used if the **CertificateName** property is also set to a valid certificate name.

If you are using a local certificate store, with the certificate and private key stored in the registry, you can explicitly specify whether the certificate store for the current user or the local machine (all users) should be used. This is done by prefixing the certificate store name with "HKCU" for the current user, or "HKLM" for the local machine. For example, a certificate store name of "HKLM:MY" would specify the personal certificate store for the local machine, rather than the current user. If neither prefix is specified, it will default to the certificate store for the current user.

This property may also be used to specify a file that contains the client certificate. In this case, the property should specify the full path to the file and must contain both the certificate and private key in PKCS12

format. If the file is protected by a password, the **CertificatePassword** property must also be set to specify the password.

See Also

HttpClient Class | SocketTools Namespace | CertificatePassword Property | Secure Property

HttpClient.CertificateSubject Property

Gets a value that provides information about the organization that the server certificate was issued to.

[Visual Basic]

```
Public ReadOnly Property CertificateSubject As String
```

[C#]

```
public string CertificateSubject {get;}
```

Property Value

A string that contains a comma separated list of name value pairs.

Remarks

The **CertificateSubject** property returns a string that contains information about the organization that the server certificate was issued to. The string value is a comma separated list of tagged name and value pairs. In the nomenclature of the X.500 standard, each of these pairs are called a relative distinguished name (RDN), and when concatenated together, forms the issuer's distinguished name (DN). For example:

```
C=US, O="RSA Data Security, Inc.", OU=Secure Server Certification Authority
```

To obtain a specific value, such as the name of the issuer or the issuer's country, the application must parse the string returned by this property. Some of the common tokens used in the distinguished name are:

Token	Description
C	The ISO standard two character country code.
S	The name of the state or province.
L	The name of the city or locality.
O	The name of the company or organization.
OU	The name of the department or organizational unit
CN	The common name; with X.509 certificates, this is the domain name of the site the certificate was issued for.

This property will return an empty string if a secure connection has not been established with the remote host.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.CipherStrength Property

Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.

[Visual Basic]

```
Public ReadOnly Property CipherStrength As Integer
```

[C#]

```
public int CipherStrength {get;}
```

Property Value

An integer value which specifies the encryption key length if a secure connection has been established; otherwise a value of 0 is returned.

Remarks

The **CipherStrength** property returns the number of bits in the key used to encrypt the secure data stream. Common values returned by this property are 128 and 256. A key length of 40 or 56 bits is considered insecure and subject to brute force attacks. 128-bit and 256-bit keys are considered secure. If this property returns a value of 0, this means that a secure connection has not been established with the remote host.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.Compression Property

Gets and sets a value that specifies if data compression should be enabled.

[Visual Basic]
```
Public Property Compression As Boolean
```

[C#]
```
public bool Compression {get; set;}
```

Property Value

Returns **true** if a data compression is enabled; otherwise returns **false**. The default value is **false.**

Remarks

The **Compression** property is used to indicate to the server whether or not it is acceptable to compress the data that is returned to the client. If compression is enabled, the client will advertise that it will accept compressed data by setting the **Accept-Encoding** request header. The server will decide whether a resource being requested can be compressed. If the data is compressed, the control will automatically expand the data before returning it to the caller.

Enabling compression does not guarantee that the data returned by the server will actually be compressed, it only informs the server that the client is willing to accept compressed data. Whether or not a particular resource is compressed depends on the server configuration, and the server may decide to only compress certain types of resources, such as text files. Disabling compression informs the server that the client is not willing to accept compressed data; this is the default.

If the **SetHeader** method is used to explicitly set the **Accept-Encoding** header to request compressed data and compression is not enabled, the class will not attempt to automatically expand the data returned by the server. In this case, the raw compressed data will be returned and the application is responsible for processing it. This behavior is by design to maintain backwards compatibility with previous versions of the control that did not have internal support for compression.

To determine if the server compressed the data returned to the client, use the **GetHeader** method to get the value of the **Content-Encoding** header. If the header is defined, the value specifies the compression method used, otherwise the data was not compressed.

This property value is only meaningful when downloading files from a server that supports file compression. It has no effect on file uploads.

See Also

HttpClient Class | SocketTools Namespace | GetData Method | GetFile Method

HttpClient.CookieCount Property

Gets the number of cookies set by the server in response to a request for a resource.

[Visual Basic]

```
Public ReadOnly Property CookieCount As Integer
```

[C#]

```
public int CookieCount {get;}
```

Property Value

An integer value which specifies the number of available cookies.

Remarks

This value can be used in conjunction with the **CookieName** and **CookieValue** properties to enumerate all of the available cookies and their values.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.Encoding Property

Gets and sets the content encoding type.

[Visual Basic]
```
Public Property Encoding As HttpEncoding
```

[C#]
```
public HttpClient.HttpEncoding Encoding {get; set;}
```

Property Value

An HttpEncoding enumeration value which specifies the current encoding type.

Remarks

The **Encoding** property explicitly sets the type of encoding used when optional parameter data is submitted with a request for a resource. By default, data is URL encoded and the content type will be designated as application/x-www-form-urlencoded.

If an application must specify its own Content-Type header, this property must be set to **HttpEncoding.encodingNone** to prevent the control from replacing the header value when the request is sent to the server. Changes to this property and any calls to the **SetHeader** method should be made after the connection to the server has been established, immediately before the resource is requested.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.FormAction Property

Gets and sets the path to the script that will accept the form data on the server.

[Visual Basic]

```
Public Property FormAction As String
```

[C#]

```
public string FormAction {get; set;}
```

Property Value

A string which specifies a script on the server.

Remarks

The **FormAction** property is used to specify the name of the script that will process the form data submitted by the control. This property is only used by the **SubmitForm** method and changing the property value does not change the current resource.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.FormMethod Property

Gets and sets the method used to submit the form data.

[Visual Basic]

```
Public Property FormMethod As HttpFormMethod
```

[C#]

```
public HttpClient.HttpFormMethod FormMethod {get; set;}
```

Property Value

An HttpFormMethod enumeration which specifies the method used to submit the form data.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.FormType Property

Gets and sets the type of encoding used to submit the form data.

[Visual Basic]

```
Public Property FormType As HttpFormType
```

[C#]

```
public HttpClient.HttpFormType FormType {get; set;}
```

Property Value

An HttpFormType enumeration value which specifies the form type and encoding method.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.Handle Property

Gets a value that specifies the client handle allocated for the current session.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer which represents a client handle. If there is no active connection, a value of -1 is returned.

Remarks

The **Handle** property specifies the numeric descriptor of the current client session.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.HashStrength Property

Gets a value which specifies the length of the message digest that was selected for a secure connection.

[Visual Basic]

```
Public ReadOnly Property HashStrength As Integer
```

[C#]

```
public int HashStrength {get;}
```

Property Value

An integer value which specifies the length of the message digest if a secure connection has been established; otherwise a value of 0 is returned.

Remarks

The **HashStrength** property returns the number of bits used in the message digest (hash) that was selected. Common values returned by this property are 128 and 160. If this property returns a value of 0, this means that a secure connection has not been established with the remote host.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.HeaderField Property

Gets and sets the name of the current header field.

[Visual Basic]
```
Public Property HeaderField As String
```

[C#]
```
public string HeaderField {get; set;}
```

Property Value

A string which specifies the current header field.

Remarks

The **HeaderField** property is used in conjunction with the **HeaderValue** property to set and/or get the values of specific fields in the HTTP request header. For example, setting this property to the value "Content-Length" and then reading the value of the **HeaderValue** property would cause the class to return length (in bytes) of the specified resource.

Note that the control automatically generates a default request header, and it is not required that the client use the **HeaderField** and **HeaderValue** properties unless it has a specific need to do so

See Also

HttpClient Class | SocketTools Namespace

HttpClient.HeaderValue Property

Gets the value of a response header field or sets the value of a request header field.

[Visual Basic]

```
Public Property HeaderValue As String
```

[C#]

```
public string HeaderValue {get; set;}
```

Property Value

A string which specifies the header field value.

Remarks

The **HeaderValue** property is used in conjunction with the **HeaderField** property to set or get the values of specific fields in the HTTP request and response headers. When the property is set to a value, then the specified request header field is set to this value. When the property is read, then the value associated with the specified response header field is returned.

Note that the control automatically generates a default request header, and it is not required that the client use the **HeaderField** and **HeaderValue** properties unless it has a specific need to do so.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.HostAddress Property

Gets and sets a value which specifies the Internet address used to establish a connection.

[Visual Basic]

```
Public Property HostAddress As String
```

[C#]

```
public string HostAddress {get; set;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **HostAddress** property can be used to set the Internet address for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the address is assigned to a valid host name, the **HostName** property will be updated with that value.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.HostName Property

Gets and sets a value which specifies the host name used to establish a connection.

[Visual Basic]

```
Public Property HostName As String
```

[C#]

```
public string HostName {get; set;}
```

Property Value

A string which specifies a host name.

Remarks

The **HostName** property can be used to set the host name for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the name can be resolved to a valid Internet address, the **HostAddress** property will be updated with that value.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.IsBlocked Property

Gets a value which indicates if the current thread is performing a blocking client operation.

[Visual Basic]

```
Public ReadOnly Property IsBlocked As Boolean
```

[C#]

```
public bool IsBlocked {get;}
```

Property Value

Returns **true** if the current thread is blocking, otherwise returns **false**.

Remarks

The **IsBlocked** property returns **true** if the current thread is blocked performing an operation. Because the Windows Sockets API only permits one blocking operation per thread of execution, this property should be checked before starting any blocking operation in response to an event.

If the **IsBlocked** property returns **false**, this means there are no blocking operations on the current thread at that time. However, this does not guarantee that the next client operation will not fail. An application should always check the return value from a client operation and check the value of the **LastError** property if an error occurs.

Note that this property will return **true** if there is any blocking operation being performed by the current thread, regardless of whether this specific instance of the class is responsible for the blocking operation or not.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.IsConnected Property

Gets a value which indicates if a connection to the remote host has been established.

[Visual Basic]
```
Public ReadOnly Property IsConnected As Boolean
```

[C#]
```
public bool IsConnected {get;}
```

Property Value

Returns **true** if the connection has been established; otherwise returns **false**.

Remarks

The **IsConnected** property can only be used to indicate if there is still a logical connection to the remote host. It cannot be used to detect abnormal conditions such as the remote host aborting the connection, the physical network connection being lost or other critical errors.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.IsInitialized Property

Gets a value which indicates if the current instance of the class has been initialized successfully.

[Visual Basic]

```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]

```
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the class instance has been initialized; otherwise returns **false**.

Remarks

The **IsInitialized** property is used to determine if the current instance of the class has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the class may not be able to initialize itself.

The most common reasons that a class instance may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required networking libraries or not being able to access the system registry.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.IsReadable Property

Gets a value which indicates if there is data available to be read from the socket connection to the server.

[Visual Basic]

```
Public ReadOnly Property IsReadable As Boolean
```

[C#]

```
public bool IsReadable {get;}
```

Property Value

Returns **true** if there is data available to be read; otherwise returns **false**.

Remarks

The **IsReadable** property returns **true** if data can be read from the client without blocking. For non-blocking sessions, this property can be checked before the application attempts to read the client. Note that even if this property does return **true** indicating that there is data available to be read, applications should always check the return value from the **Read** method.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.IsWritable Property

Gets a value which indicates if data can be written to the client without blocking.

[Visual Basic]

```
Public ReadOnly Property IsWritable As Boolean
```

[C#]

```
public bool IsWritable {get;}
```

Property Value

Returns **true** if data can be written to the client; otherwise returns **false**.

Remarks

The **IsWritable** property returns **true** if data can be written to the client without blocking. For non-blocking sessions, this property can be checked before the application attempts to write data to the client. Note that even if this property does return **true** indicating that data can be written to the client, applications should always check the return value from the **Write** method.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.KeepAlive Property

Gets and sets a Boolean value that specifies if the connection to the server is persistent.

[Visual Basic]
```
Public Property KeepAlive As Boolean
```

[C#]
```
public bool KeepAlive {get; set;}
```

Property Value

A boolean value that specifies if the connection to the server is persistent.

Remarks

Setting the **KeepAlive** property to a value of **true** indicates that the client wishes to maintain a persistent connection with the server. For those clients who wish to retrieve a number of documents, this is more efficient because the client does not need to connect, retrieve the document and disconnect each time. Instead, the client can connect, retrieve each document and then disconnect when it is finished. If the property value is **false**, a persistent connection is not maintained, and the client must establish a connection for each document that it wishes to retrieve. This property should be set to the desired value before establishing a connection with the remote server.

Note that this option is only available for those servers which support version 1.0 or later of the HTTP protocol. For version 1.0 servers, the connection header field is set to the value 'keep-alive', which instructs compliant servers to maintain a persistent connection. For version 1.1 and later, persistent connections are the default. In this case, if the property value is set to **false**, the connection header field will be set to the value 'close', telling the server that you wish to close the connection after the document has been retrieved. It is possible that the server may choose to close the connection itself, even if it supports persistent connections. If the server does not support persistent connections and the **KeepAlive** property is set to **true**, the client will attempt to simulate a persistent connection by automatically reconnecting for each request.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public HttpClient.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.LocalAddress Property

Gets the local Internet address that the client is bound to.

[Visual Basic]

```
Public ReadOnly Property LocalAddress As String
```

[C#]

```
public string LocalAddress {get;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **LocalAddress** property returns the local Internet address that the client is bound to when a connection is established with a remote host. This property may return either an IPv4 or IPv6 formatted address, depending on the type of connection that was established.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.Localize Property

Gets and sets a value which specifies if date values are localized to the current timezone.

[Visual Basic]
```
Public Property Localize As Boolean
```

[C#]
```
public bool Localize {get; set;}
```

Property Value

A boolean value which specifies if date values are localized to the current timezone.

Remarks

Setting the **Localize** property controls how remote file date and time values are localized when the **GetFileTime** method is called. If the property is set to **true**, then the file date and time will be adjusted to the current timezone. If the property is set to **false**, which is the default value, then the file date and time are returned as UTC (Coordinated Universal Time) values.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.LocalName Property

Gets a value which specifies the host name for the local system.

[Visual Basic]

```
Public ReadOnly Property LocalName As String
```

[C#]

```
public string LocalName {get;}
```

Property Value

A string which specifies the local host name.

Remarks

The **LocalName** property returns the fully-qualified host name assigned to the local system. If the system has not been configured with an Internet domain name, then this property will return the NetBIOS name assigned to the local system.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.LocalPort Property

Gets the local port number the client is bound to.

[Visual Basic]

```
Public ReadOnly Property LocalPort As Integer
```

[C#]

```
public int LocalPort {get;}
```

Property Value

An integer value which specifies a port number. The default value is 0.

Remarks

The **LocalPort** property is used to identify the local port number that the client is bound to to when a connection is established with a remote host.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.Options Property

Gets and sets a value which specifies one or more client options.

[Visual Basic]

```
Public Property Options As HttpOptions
```

[C#]

```
public HttpClient.HttpOptions Options {get; set;}
```

Property Value

Returns one or more HttpOptions enumeration flags which specify the options for the client. The default value for this property is **httpOptionNone**.

Remarks

The **Options** property specifies one or more default options options which are used when establishing a connection using the **Connect** method.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.Password Property

Gets and sets the password used to authenticate the client session.

[Visual Basic]

```
Public Property Password As String
```

[C#]

```
public string Password {get; set;}
```

Property Value

A string which specifies the password.

Remarks

The **Password** property specifies the password used to authenticate the client session. This property is used as the default value for the **Connect** method if no password is specified as an argument.

Refer to the **Authentication** property for more information on the available authentication methods. If you are using the **authBearer** authentication method, this property should not be set to the user's password. Instead, you should set the **BearerToken** property to the access token issued by the mail service provider. Note that these tokens can be much larger than your typical password and are only valid for a limited period of time.

You can use the **Password** property to specify a bearer token. However, it is recommended that you use the **BearerToken** property instead of assigning it to this property. It will ensure compatibility with future versions of the class and make it clear in your code you are using an OAuth 2.0 bearer token and not a password. If the **Authentication** property specifies the **authBearer** authentication method, this property will return the bearer token.

See Also

HttpClient Class | SocketTools Namespace | Authentication Property | BearerToken Property | UserName Property | Connect Method

HttpClient.Priority Property

Gets and sets a value which specifies the priority of data transfers.

[Visual Basic]

```
Public Property Priority As HttpPriority
```

[C#]

```
public HttpClient.HttpPriority Priority {get; set;}
```

Property Value

Returns a HttpPriority enumeration value which specify the current data transfer priority. The default value for this property is **priorityNormal**.

Remarks

The **Priority** property can be used to control the processor usage, memory and network bandwidth allocated for data transfers. The default priority balances resource utilization and transfer speed while ensuring that a single-threaded application remains responsive to the user. Lower priorities reduce the overall resource utilization at the expense of transfer speed. For example, if you create a worker thread to download a file in the background and want to ensure that it has a minimal impact on the process, the **priorityBackground** value can be used.

Higher priority values increase the memory allocated for the transfers and increases processor utilization for the transfer. The **priorityCritical** priority maximizes transfer speed at the expense of system resources. It is not recommended that you increase the data transfer priority unless you understand the implications of doing so and have thoroughly tested your application. If the data transfer is being performed in the main UI thread, increasing the priority may interfere with the normal processing of Windows messages and cause the application to appear to become non-responsive. It is also important to note that when the priority is set to **priorityCritical**, normal progress events will not be generated during the transfer.

See Also

HttpClient Class | SocketTools Namespace | HttpPriority Enumeration

HttpClient.ProtocolVersion Property

Gets and sets a value which specifies the default protocol version.

[Visual Basic]

```
Public Property ProtocolVersion As HttpVersion
```

[C#]

```
public HttpClient.HttpVersion ProtocolVersion {get; set;}
```

Property Value

An HttpVersion enumeration which specifies the protocol version.

Remarks

The **ProtocolVersion** property sets or returns the current HTTP version number. It is used to determine how requests are submitted to the server, as well as what header fields are required. The default value for this property is **HttpVersion.version10**, and should be changed before any connection attempt is made by the client.

Note that setting the property value to **HttpVersion.version09** tells the client to use the preliminary protocol specification which only supported a basic version of the GET command, and did not have any provisions for features such as user authentication, virtual hosting, etc. Header fields are not supported in this version of the protocol.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.ProxyHost Property

Gets and sets the hostname or IP address of a proxy server.

[Visual Basic]

```
Public Property ProxyHost As String
```

[C#]

```
public string ProxyHost {get; set;}
```

Property Value

A string which specifies the hostname or IP address of the proxy server that will be used when establishing a connection.

Remarks

The **ProxyHost** property should be set to the name of the proxy server that you want to connect to. This property may be set to either a fully qualified domain name, or an IP address. This property is only used if the **ProxyType** property specifies a proxy server type other than **proxyNone**.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.ProxyPassword Property

Gets and sets the password used to authenticate the connection to a proxy server.

[Visual Basic]

```
Public Property ProxyPassword As String
```

[C#]

```
public string ProxyPassword {get; set;}
```

Property Value

A string which specifies a password.

Remarks

The **ProxyPassword** property specifies the password used to authenticate the user to the proxy server. If a password is not required by the server, this property is ignored.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.ProxyPort Property

Gets and sets a value that specifies the proxy server port number.

[Visual Basic]

```
Public Property ProxyPort As Integer
```

[C#]

```
public int ProxyPort {get; set;}
```

Property Value

An integer value which specifies the proxy port number.

Remarks

The **ProxyPort** property is used to set the port number that the control will use to establish a connection with the proxy server. A value of zero specifies that the client will connect to the proxy server using the standard HTTP service port.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.ProxyType Property

Gets and sets the type of proxy server the client will use to establish a connection.

[Visual Basic]

```
Public Property ProxyType As HttpProxyType
```

[C#]

```
public HttpClient.HttpProxyType ProxyType {get; set;}
```

Property Value

An HttpProxyType enumeration which specifies the type of proxy that the client will connect through.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.ProxyUser Property

Gets and sets the username used to authenticate the connection to a proxy server.

[Visual Basic]

```
Public Property ProxyUser As String
```

[C#]

```
public string ProxyUser {get; set;}
```

Property Value

A string which specifies the username.

Remarks

The **ProxyUser** property specifies the user that is logging in to the proxy server. If the proxy server does not require the user to login, then this property is ignored.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.RemotePort Property

Gets and sets a value which specifies the remote port number.

[Visual Basic]

```
Public Property RemotePort As Integer
```

[C#]

```
public int RemotePort {get; set;}
```

Property Value

An integer value which specifies a port number.

Remarks

The **RemotePort** property is used to set the port number that will be used to establish a connection with a remote host. If the port number specifies a well-known port, the **RemoteService** property will be updated with that name.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.Resource Property

Gets and sets a value which specifies a resource on the server.

[Visual Basic]
```
Public Property Resource As String
```

[C#]
```
public string Resource {get; set;}
```

Property Value

A string which specifies a resource.

Remarks

The **Resource** property is used to specify the name of a resource on the server. The resource may be a file, such as an HTML document or an image, or it may be a script used to process data submitted by the client. Note that this property specifies the name of the resource only, not a complete URL. To specify a complete URL, set the **URL** property and the control will automatically set the **Resource** property to the correct value.

In most cases, the resource name should be specified using an absolute path that begins with a leading slash character.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.ResultCode Property

Gets a value which specifies the last result code returned by the server.

[Visual Basic]
```
Public ReadOnly Property ResultCode As Integer
```

[C#]
```
public int ResultCode {get;}
```

Property Value

An integer value which specifies the last result code returned by the server.

Remarks

Result codes are three-digit numeric values returned by the remote server and may be broken down into the following ranges:

ResultCode	Description
100-199	Positive preliminary result. This indicates that the requested action is being initiated, and the client should expect another reply from the server before proceeding.
200-299	Positive completion result. This indicates that the server has successfully completed the requested action.
300-399	Positive intermediate result. This indicates that the requested action cannot complete until additional information is provided to the server.
400-499	Transient negative completion result. This indicates that the requested action did not take place, but the error condition is temporary and may be attempted again.
500-599	Permanent negative completion result. This indicates that the requested action did not take place.

It is important to note that while some result codes have become standardized, not all servers respond to commands using the same result codes. It is recommended that applications check for ranges of values to determine if a command was successful, not a specific value.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.ResultString Property

Gets a string value which describes the result of the previous command.

[Visual Basic]

```
Public ReadOnly Property ResultString As String
```

[C#]

```
public string ResultString {get;}
```

Property Value

A string which describes the result of the previous command executed on the server.

Remarks

The **ResultString** property returns the result string from the last action taken by the client. This string is generated by the remote server, and typically is used to describe the result code. For example, if an error is indicated by the result code, the result string may describe the condition that caused the error.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.Secure Property

Gets and sets a value which specifies if a secure connection is established.

[Visual Basic]

```
Public Property Secure As Boolean
```

[C#]

```
public bool Secure {get; set;}
```

Property Value

Returns **true** if a secure connection is established; otherwise returns **false**. The default value is **false.**

Remarks

The **Secure** property determines if a secure connection is established with the remote host. The default value for this property is **false**, which specifies that a standard connection to the server is used. To establish a secure connection, the application should set this property value to **true** prior to calling the **Connect** method. Once the connection has been established, the client may exchange data with the server as with standard connections.

It is strongly recommended that any application that sets this property **true** use error handling to trap an errors that may occur. If the control is unable to initialize the security libraries, or otherwise cannot create a secure session for the client, an exception may be generated when this property value is set.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.SecureCipher Property

Gets a value that specifies the encryption algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureCipher As SecureCipherAlgorithm
```

[C#]

```
public HttpClient.SecureCipherAlgorithm SecureCipher {get;}
```

Property Value

A SecureCipherAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureCipher** property returns a value which identifies the algorithm used to encrypt the data stream. If a secure connection has not been established, this property will return a value of **cipherNone**.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.SecureHash Property

Gets a value that specifies the message digest algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureHash As SecureHashAlgorithm
```

[C#]

```
public HttpClient.SecureHashAlgorithm SecureHash {get;}
```

Property Value

A SecureHashAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureHash** property returns a value which identifies the message digest (hash) algorithm that was selected when a secure connection was established. If a secure connection has not been established, this property will return a value of **hashNone**.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.SecureKeyExchange Property

Gets a value that specifies the key exchange algorithm used for a secure connection.

[Visual Basic]
```
Public ReadOnly Property SecureKeyExchange As SecureKeyAlgorithm
```

[C#]
```
public HttpClient.SecureKeyAlgorithm SecureKeyExchange {get;}
```

Property Value

A SecureKeyAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureKeyExchange** property returns a value which identifies the key exchange algorithm that was selected when a secure connection was established. If a secure connection has not been established, this property will return a value of **keyExchangeNone**.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.SecureProtocol Property

Gets and sets a value which specifies the protocol used for a secure connection.

[Visual Basic]

```
Public Property SecureProtocol As SecurityProtocols
```

[C#]

```
public HttpClient.SecurityProtocols SecureProtocol {get; set;}
```

Property Value

A SecurityProtocols enumeration value which identifies the protocol to be used when establishing a secure connection.

Remarks

The **SecureProtocol** property can be used to specify the security protocol to be used when establishing a secure connection with a server or accepting a secure connection from a client. By default, the class will attempt to use either SSL v3 or TLS v1 to establish the connection, with the appropriate protocol automatically selected based on the capabilities of the remote host. It is recommended that you only change this property value if you fully understand the implications of doing so. Assigning a value to this property will override the default protocol and force the class to attempt to use only the protocol specified.

Multiple security protocols may be specified by combining them using a bitwise **or** operator. After a connection has been established, this property will identify the protocol that was selected. Attempting to set this property after a connection has been established will result in an exception being thrown. This property should only be set after setting the **Secure** property to **true** and before calling the **Accept** or **Connect** methods.

In some cases, a server may only accept a secure connection if the TLS v1 protocol is specified. If the security protocol is not compatible with the server, then the connection will fail with an error indicating that the control is unable to establish a security context for the session. In this case, try assigning the property to **protocolTLS1** and attempt the connection again.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.Status Property

Gets a value which specifies the current status of the client.

[Visual Basic]

```
Public ReadOnly Property Status As HttpStatus
```

[C#]

```
public HttpClient.HttpStatus Status {get;}
```

Property Value

A HttpStatus enumeration value which specifies the current client status.

Remarks

The **Status** property returns the current status of the client. This property can be used to check on blocking connections to determine if the client is interacting with the remote host before taking some action.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.TaskCount Property

Get the number of active background file transfers.

[Visual Basic]
```
Public ReadOnly Property TaskCount As Integer
```

[C#]
```
public int TaskCount {get;}
```

Property Value

An integer value that specifies the number of background file transfers that are currently in progress.

Remarks

The **TaskCount** property returns the number of background file transfers that are currently in progress. One common use for this property is to create a timer that periodically checks this value when a series of background transfers are started. When the property returns a value of zero, that indicates all of the background transfers have completed. This property can also be used to enumerate the active background tasks in conjunction with the **TaskList** property.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.TaskId Property

Get the task identifier for the last background file transfer.

[Visual Basic]

```
Public ReadOnly Property TaskId As Integer
```

[C#]

```
public int TaskId {get;}
```

Property Value

An integer value the uniquely identifies the current background task.

Remarks

The **TaskId** property returns the task ID associated with the current background task. This identifies the last background file transfer that was initiated with a call to the **AsyncGetFile** or **AsyncPutFile** methods. This property value will change with each subsequent background transfer that is performed. If this property returns a value of zero, that indicates that no background tasks have been started for this instance of the class.

To enumerate the active background tasks, use the **TaskCount** property and the **TaskList** array.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.TaskList Property

Get an array of active background task identifiers.

[Visual Basic]

```
Public ReadOnly Property TaskList As ArrayList
```

[C#]

```
public System.Collections.ArrayList TaskList {get;}
```

Property Value

An **ArrayList** object that contains a list of integer values that uniquely identify the active background tasks that have been started by this instance of the class.

Remarks

The **TaskList** property returns a read-only **ArrayList** object that is popularted with the task identifiers for all active background tasks that have been created by this instance of the class. The current number of active tasks can be determined using the **TaskCount** property.

As background tasks complete and additional tasks are started, the values stored in this array will change. The application should never make any assumptions about the numeric values stored in the array or the order they are returned. Task IDs should be considered opaque values that are unique to the process. When a background task completes, its corresponding ID is removed from the list of active tasks and this can potentially change the task ID values associated with each index into the array.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.ThreadModel Property

Gets and sets a value which specifies the threading model for the class instance.

[Visual Basic]
```
Public Property ThreadModel As ThreadingModel
```

[C#]
```
public HttpClient.ThreadingModel ThreadModel {get; set;}
```

Property Value

Returns one or more ThreadingModel enumeration value which specifies the threading model for the client. The default value for this property is **modelSingleThread**.

Remarks

The **ThreadModel** property specifies the threading model that is used by the class instance when a connection is established. The default value for this property is **modelSingleThread**, which specifies that only the thread that established the connection should be permitted to invoke methods. It is important to note that this threading model does not limit the application to a single thread of execution. When a session is established using the **Connect** method, that session is attached to the thread that created it. From that point on, until the session is terminated, only the owner may invoke methods in that instance of the class. The ownership of the class instance may be transferred from one thread to another using the **AttachThread** method.

Setting this property to **modelFreeThread** disables certain internal safety checks that are performed by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads. The application must ensure that no two threads will attempt to invoke a blocking method at the same time. In other words, if one thread invokes a method, the application must ensure that another thread will not attempt to invoke any other method at the same time using the same instance of the class.

Changing the value of this property will not affect an active client session. The threading model must be specified prior to invoking the **Connect** method.

See Also

HttpClient Class | SocketTools Namespace | AttachThread Method | ThreadingModel Enumeration | ThreadModel Attribute

HttpClient.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]

```
Public Property ThrowError As Boolean
```

[C#]

```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.Timeout Property

Gets and sets a value which specifies a timeout period in seconds.

[Visual Basic]
```
Public Property Timeout As Integer
```

[C#]
```
public int Timeout {get; set;}
```

Property Value

An integer value which specifies a timeout period in seconds.

Remarks

Setting the **Timeout** property specifies the number of seconds until a blocking operation fails and returns an error.

The timeout period is only used when the client is in blocking mode. Although this property can be changed when the client is in non-blocking mode, the value will be ignored until the client is returned to blocking mode.

For most applications it is recommended the timeout period be set between 10 and 20 seconds.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.TimeZone Property

Gets and sets the current timezone offset in seconds.

[Visual Basic]

```
Public Property TimeZone As Integer
```

[C#]

```
public int TimeZone {get; set;}
```

Property Value

An integer value which specifies the current timezone offset in seconds.

Remarks

The **TimeZone** property returns the current offset from UTC in seconds. Setting the property changes the current timezone offset to the specified value. The value of this property is initially determined by the date and time settings on the local system.

This property value is used in conjunction with the **Localize** property to control how date and time localization is handled.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.Trace Property

Gets and sets a value which indicates if network function logging is enabled.

[Visual Basic]

```
Public Property Trace As Boolean
```

[C#]

```
public bool Trace {get; set;}
```

Property Value

Returns **true** if network function tracing is enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Trace** property is used to enable (or disable) the tracing of network function calls. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the SocketTools networking classes and is enabled or disabled for an entire process. This means that once trace logging is enabled for a given component, all of the function calls made by the process using any of the SocketTools classes will be logged. For example, if you have an application using both the File Transfer Protocol and Post Office Protocol classes, and you set the **Trace** property to **true**, function calls made by both classes will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all classes used by the process.

If trace logging is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the logfile is fairly large. Since logfiles can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

When redistributing your application, make sure that you include the **SocketTools10.TraceLog.dll** module with your installation. If this library is not present, then no trace output will be generated and the value of the **Trace** property will be ignored. Only those function calls made by the SocketTools networking classes will be logged. Calls made directly to the Windows Sockets API, or calls made by other classes, will not be logged.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.TraceFile Property

Gets and sets a value which specifies the name of the logfile.

[Visual Basic]

```
Public Property TraceFile As String
```

[C#]

```
public string TraceFile {get; set;}
```

Property Value

A string which specifies the name of the file.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named **SocketTools.log** is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFile** property should be set to the same value for each instance. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
 MyApp INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0 MyApp WRN:
connect(46, 192.0.0.1:1234, 16) returned -1 [10035] MyApp ERR: accept(46,
NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the control is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Note that if the specified file cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.TraceFlags Property

Gets and sets a value which specifies the client function tracing flags.

[Visual Basic]

```
Public Property TraceFlags As TraceOptions
```

[C#]

```
public HttpClient.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration which specifies the amount of detail written to the trace logfile.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Because network function tracing is enabled per-process, the trace flags are shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFlags** property should be set to the same value for each instance. Changing the trace flags for any one instance of the class will affect the logging performed for all SocketTools classes used by the application.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being written and the error **errorOperationWouldBlock** occurs, a warning is generated because the application simply needs to attempt to write the data at a later time.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.URL Property

Gets and sets the current URL used to access a file on the server.

[Visual Basic]
```
Public Property URL As String
```

[C#]
```
public string URL {get; set;}
```

Property Value

A string which specifies the current URL.

Remarks

The **URL** property returns the current Uniform Resource Locator string which is used by the control to access a resource on the server. URLs have a specific format which provides information about the remote host, port, resource, as well as optional information such as a username and password for authentication:

```
    http://[username : [password] @] remotehost [:remoteport] / resource
[? parameters]
```

The first part of the URL is the protocol and in this case will always be "http", or "https" if a secure connection is being used. If a username and password is required for authentication, then this will be included in the URL before the name of the remote host. Next, there is the name of the remote host to connect to, optionally followed by a port number. If no port number is given, then the default port for the protocol will be used. This is followed by the resource, which is usually a path to a file or script on the server. Parameters to the resource may also be specified, which are typically used as arguments to a script that is executed on the server.

Here are some common examples of URLs used to access resources on an HTTP server:

```
http://www.example.com/products/index.html
```

In this example, the remote host is www.example.com and the resource is /products/index.html. The default port will be used to access the resource, and no username and password is provided for authentication.

```
http://www.example.com:8080/index.html
```

In this example, the remote host is www.example.com and the resource is /products/index.html. However, the client should connect to an alternative port number, in this case 8080.

```
https://www.example.com/order/confirm.asp
```

In this example, the remote host is www.example.com and the resource is the script /order/confirm.asp. Because the protocol is https, a secure connection on port 443 will be established.

```
http://jsmith:secret@www.example.com:8080/~jsmith/personal/index.html
```

In this example, the remote host is www.example.com and the resource is /~jsmith/personal/index.html. The port 8080 will be used to access the resource, and access to the resource will be authenticated with the username "jsmith" and the password "secret".

When setting the **URL** property, the control will parse the string and automatically update the **HostName**, **RemotePort**, **UserName**, **Password** and **Resource** properties according to the values specified in the URL. This enables an application to simply provide the URL and then call the **Connect** method to establish the connection.

Note that if this property is assigned a value which cannot be parsed, the control will throw an error that indicates that the property value is invalid. In a language like Visual Basic it is important that you implement an error handler, particularly if you are assigning a value to the property based on user input. If the user enters an invalid URL and there is no error handler, it could result in an exception which terminates the application.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.UserName Property

Gets and sets the username used to authenticate the client session.

[Visual Basic]

```
Public Property UserName As String
```

[C#]

```
public string UserName {get; set;}
```

Property Value

A string which specifies the username.

Remarks

If a username is not specified when the **Connect** method is called, the value of this property will be used as the default username when establishing a connection with the server.

See Also

HttpClient Class | SocketTools Namespace | Authentication Property | BearerToken Property | Password Property | Connect Method

HttpClient.Version Property

Gets a value which returns the current version of the HttpClient class library.

[Visual Basic]

```
Public ReadOnly Property Version As String
```

[C#]

```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the HttpClient class library. This value can be used by an application for validation and debugging purposes.

See Also

HttpClient Class | SocketTools Namespace

HttpClient Methods

The methods of the **HttpClient** class are listed below. For a complete list of **HttpClient** class members, see the HttpClient Members topic.

Public Static (Shared) Methods

ErrorText	Returns the description of an error code.

Public Instance Methods

AddField	Overloaded. Add the form field and its value to the current form.
AddFile	Append the contents of the file to the current form.
AsyncGetFile	Overloaded. Download a file from the server to the local system in the background.
AsyncPutFile	Overloaded. Upload a file from the local system to the server in the background.
AttachThread	Attach an instance of the class to the current thread
Authenticate	Overloaded. Authenticate the client session with a username and password.
Cancel	Cancel the current blocking client operation.
ClearForm	Remove all defined fields from the current form.
ClearHeaders	Clears the current request and response headers.
CloseFile	Close the file that was opened on the server.
Command	Overloaded. Send a custom command to the web server.
Connect	Overloaded. Establish a connection with a remote host.
CreateFile	Overloaded. Create a new file or overwrite an existing file on the web server.
CreateForm	Overloaded. Create a new virtual form.
DeleteField	Delete a form field and its value from the current form.
DeleteFile	Remove a file on the web server.
Disconnect	Terminate the connection with a remote host.
Dispose	Overloaded. Releases all resources used by HttpClient.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetCookie	Overloaded. Return information about the

	specified cookie.
GetData	Overloaded. Transfer data from the web server and store it in a local buffer.
GetFile	Overloaded. Transfer data from the web server and store it in a file on the local system.
GetFileSize	Overloaded. Return the size of the specified file on the web server.
GetFileTime	Overloaded. Return the modification date and time for specified file on the web server.
GetFirstHeader	Return the first response header field name and value.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetHeader	Return the value of the specified response header field.
GetNextHeader	Return the next response header field name and value.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the HttpClient class.
OpenFile	Open a file on the web server for reading.
PatchData	Overloaded. Submits patch data to the server and returns the result in a string.
PostData	Overloaded. Submits the contents of the specified buffer to a resource on the server.
PostFile	Overloaded. Upload the contents of a file to a resource on the server.
PostJson	Overloaded. Submits JSON formatted data to the server and returns the result in a string.
PostXml	Overloaded. Submits XML formatted data to the server and returns the result in a string.
PutData	Overloaded. Transfer data from a local buffer to the server.
PutFile	Overloaded. Transfer a file from the local system to the web server.
Read	Overloaded. Read data from the server and store it in a byte array.
Reset	Reset the internal state of the object, resetting all properties to their default values.
SetCookie	Send the specified cookie to the server when a resource is requested.

SetHeader	Set the value of a request header field.
SubmitForm	Overloaded. Submits the current form data to the server for processing.
TaskAbort	Overloaded. Abort the specified asynchronous task.
TaskDone	Overloaded. Determine if an asynchronous task has completed.
TaskResume	Overloaded. Resume execution of an asynchronous task.
TaskSuspend	Overloaded. Suspend execution of an asynchronous task.
TaskWait	Overloaded. Wait for an asynchronous task to complete.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
Write	Overloaded. Write one or more bytes of data to the server.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the HttpClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.AddField Method

Add the form field and its value to the current form.

Overload List

Add the form field and its value to the current form.

public bool AddField(string,byte[],int);

Add the form field and its value to the current form.

public bool AddField(string,string);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.AddField Method (String, Byte[], Int32)

Add the form field and its value to the current form.

[Visual Basic]

```
Overloads Public Function AddField( _
   ByVal fieldName As String, _
   ByVal fieldData As Byte(), _
   ByVal fieldLength As Integer _
) As Boolean
```

[C#]

```
public bool AddField(
   string fieldName,
   byte[] fieldData,
   int fieldLength
);
```

Parameters

fieldName

A string which specifies the name of the field to add to the form.

fieldData

A byte array which specifies the data for the form field.

fieldLength

An integer value which specifies the length of the field data in bytes. This value cannot be larger than the size of the byte array passed to this method.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AddField** method is used to add a field and its associated value to a form created using the **CreateForm** method. If the field name has already been added to the form, the previous value is deleted and replaced by the new value.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.AddField Overload List

HttpClient.AddField Method (String, String)

Add the form field and its value to the current form.

[Visual Basic]

```
Overloads Public Function AddField( _
   ByVal fieldName As String, _
   ByVal fieldData As String _
) As Boolean
```

[C#]

```
public bool AddField(
   string fieldName,
   string fieldData
);
```

Parameters

fieldName
A string which specifies the name of the field to add to the form.

fieldData
A string which specifies the data for the form field.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AddField** method is used to add a field and its associated value to a form created using the **CreateForm** method. If the field name has already been added to the form, the previous value is deleted and replaced by the new value.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.AddField Overload List

HttpClient.AddFile Method

Append the contents of the file to the current form.

[Visual Basic]

```
Public Function AddFile( _
   ByVal fieldName As String, _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool AddFile(
   string fieldName,
   string fileName
);
```

Parameters

fieldName
A string which specifies the name of the field to add to the form.

fileName
A string which specifies the name of the file.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AddFile** method is used to add the contents of a file to a form created using the **CreateForm** method. If the field name has already been added to the form, the previous value is deleted and replaced by the new value.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.AsyncGetFile Method

Download a file from the server to the local system in the background.

Overload List

Download a file from the server to the local system in the background.

public bool AsyncGetFile(string);

Download a file from the server to the local system in the background.

public bool AsyncGetFile(string,string);

Download a file from the server to the local system in the background.

public bool AsyncGetFile(string,string,HttpTransferOptions);

Download a file from the server to the local system in the background.

public bool AsyncGetFile(string,string,HttpTransferOptions,long);

See Also

HttpClient Class | SocketTools Namespace | AsyncPutFile Method

HttpClient.AsyncGetFile Method (String)

Download a file from the server to the local system in the background.

[Visual Basic]

```
Overloads Public Function AsyncGetFile( _
   ByVal localFile As String _
) As Boolean
```

[C#]

```
public bool AsyncGetFile(
   string localFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be created, overwritten or appended to. The file pathing and name conventions must be that of the local host.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncGetFile** method will download the contents of a remote file to a file on the local system. It is similar to the **GetFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being downloaded from the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create a new instance of this class for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is downloaded, the class will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to download more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.AsyncGetFile Overload List | AsyncPutFile Method

HttpClient.AsyncGetFile Method (String, String)

Download a file from the server to the local system in the background.

[Visual Basic]

```
Overloads Public Function AsyncGetFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String _
) As Boolean
```

[C#]

```
public bool AsyncGetFile(
   string localFile,
   string remoteFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be created, overwritten or appended to. The file pathing and name conventions must be that of the local host.

remoteFile

A string that specifies the file on the server that will be transferred to the local system. The file pathing and name conventions must be that of the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncGetFile** method will download the contents of a remote file to a file on the local system. It is similar to the **GetFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being downloaded from the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create a new instance of this class for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is downloaded, the class will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to download more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.AsyncGetFile Overload List | AsyncPutFile Method

HttpClient.AsyncGetFile Method (String, String, HttpTransferOptions)

Download a file from the server to the local system in the background.

[Visual Basic]

```
Overloads Public Function AsyncGetFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal options As HttpTransferOptions _
) As Boolean
```

[C#]

```
public bool AsyncGetFile(
   string localFile,
   string remoteFile,
   HttpTransferOptions options
);
```

Parameters

localFile

A string that specifies the file on the local system that will be created, overwritten or appended to. The file pathing and name conventions must be that of the local host.

remoteFile

A string that specifies the file on the server that will be transferred to the local system. The file pathing and name conventions must be that of the server.

options

An HttpTransferOptions enumeration value which specifies one or more file transfer options.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncGetFile** method will download the contents of a remote file to a file on the local system. It is similar to the **GetFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being downloaded from the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create a new instance of this class for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is downloaded, the class will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to download more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.AsyncGetFile Overload List | AsyncPutFile Method

HttpClient.AsyncGetFile Method (String, String, HttpTransferOptions, Int64)

Download a file from the server to the local system in the background.

[Visual Basic]

```
Overloads Public Function AsyncGetFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal options As HttpTransferOptions, _
   ByVal offset As Long _
) As Boolean
```

[C#]

```
public bool AsyncGetFile(
   string localFile,
   string remoteFile,
   HttpTransferOptions options,
   long offset
);
```

Parameters

localFile
A string that specifies the file on the local system that will be created, overwritten or appended to. The file pathing and name conventions must be that of the local host.

remoteFile
A string that specifies the file on the server that will be transferred to the local system. The file pathing and name conventions must be that of the server.

options
An HttpTransferOptions enumeration value which specifies one or more file transfer options.

offset
A byte offset which specifies where the file transfer should begin. The default value of zero specifies that the file transfer should start at the beginning of the file. A value greater than zero is typically used to restart a transfer that has not completed successfully. Note that specifying a non-zero offset requires that the server support the REST command to restart transfers.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncGetFile** method will download the contents of a remote file to a file on the local system. It is similar to the **GetFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being downloaded from the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create a new instance of this class for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is downloaded, the class will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to download more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.AsyncGetFile Overload List | AsyncPutFile Method

HttpClient.AsyncPutFile Method

Upload a file from the local system to the server in the background.

Overload List

Upload a file from the local system to the server in the background.

public bool AsyncPutFile(string);

Upload a file from the local system to the server in the background.

public bool AsyncPutFile(string,string);

Upload a file from the local system to the server in the background.

public bool AsyncPutFile(string,string,HttpTransferOptions);

Upload a file from the local system to the server in the background.

public bool AsyncPutFile(string,string,HttpTransferOptions,long);

See Also

HttpClient Class | SocketTools Namespace | AsyncGetFile Method

HttpClient.AsyncPutFile Method (String)

Upload a file from the local system to the server in the background.

[Visual Basic]

```
Overloads Public Function AsyncPutFile( _
   ByVal localFile As String _
) As Boolean
```

[C#]

```
public bool AsyncPutFile(
   string localFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncPutFile** method will upload the contents of a file on the local system to the server. It is similar to the **PutFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being uploaded to the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create an instance of the control for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is uploaded, the control will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to upload more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.AsyncPutFile Overload List | AsyncGetFile Method

HttpClient.AsyncPutFile Method (String, String)

Upload a file from the local system to the server in the background.

[Visual Basic]

```
Overloads Public Function AsyncPutFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String _
) As Boolean
```

[C#]

```
public bool AsyncPutFile(
   string localFile,
   string remoteFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

remoteFile

A string that specifies the file on the server that will be created, overwritten or appended to. The file pathing and name conventions must be that of the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncPutFile** method will upload the contents of a file on the local system to the server. It is similar to the **PutFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being uploaded to the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create an instance of the control for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is uploaded, the control will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to upload more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.AsyncPutFile Overload List | AsyncGetFile Method

HttpClient.AsyncPutFile Method (String, String, HttpTransferOptions)

Upload a file from the local system to the server in the background.

[Visual Basic]

```
Overloads Public Function AsyncPutFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal options As HttpTransferOptions _
) As Boolean
```

[C#]

```
public bool AsyncPutFile(
   string localFile,
   string remoteFile,
   HttpTransferOptions options
);
```

Parameters

localFile
A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

remoteFile
A string that specifies the file on the server that will be created, overwritten or appended to. The file pathing and name conventions must be that of the server.

options
An HttpTransferOptions enumeration value which specifies one or more file transfer options.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncPutFile** method will upload the contents of a file on the local system to the server. It is similar to the **PutFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being uploaded to the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create an instance of the control for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is uploaded, the control will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to upload more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.AsyncPutFile Overload List | AsyncGetFile Method

HttpClient.AsyncPutFile Method (String, String, HttpTransferOptions, Int64)

Upload a file from the local system to the server in the background.

```
[Visual Basic]
Overloads Public Function AsyncPutFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal options As HttpTransferOptions, _
   ByVal offset As Long _
) As Boolean
```

```
[C#]
public bool AsyncPutFile(
   string localFile,
   string remoteFile,
   HttpTransferOptions options,
   long offset
);
```

Parameters

localFile
A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

remoteFile
A string that specifies the file on the server that will be created, overwritten or appended to. The file pathing and name conventions must be that of the server.

options
An HttpTransferOptions enumeration value which specifies one or more file transfer options.

offset
A byte offset which specifies where the file transfer should begin. The default value of zero specifies that the file transfer should start at the beginning of the file. A value greater than zero is typically used to restart a transfer that has not completed successfully. Note that specifying a non-zero offset requires that the server support the REST command to restart transfers.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AsyncPutFile** method will upload the contents of a file on the local system to the server. It is similar to the **PutFile** method, however it retrieves the file using a background worker thread and does not block the current working thread. This enables the application to continue to perform other operations while the file is being uploaded to the server. This method requires that you explicitly establish a connection using the **Connect** method. All background tasks will duplicate the active connection and use it establish a secondary connection with the server to perform the file transfer. If you wish to perform multiple asynchronous file transfers from different servers, you must create an instance of the control for each server.

After this method is called, the **OnTaskBegin** event will be fired, indicating that the background task has begun the process of connecting to the server and performing the file transfer. As the file is uploaded, the control will periodically invoke the **OnTaskRun** event handler. When the transfer has completed, the **OnTaskEnd** event will be fired. It is not required that you implement handlers for these events.

To determine when a transfer has completed without implementing any event handlers, periodically call the **TaskDone** method. If you wish to block the current thread and wait for the transfer to complete, call the **TaskWait** method. To stop a background file transfer that is in progress, call the **TaskAbort** method. This will signal the background worker thread to cancel the transfer and terminate the session.

This method can be called multiple times to upload more than one file in the background; however, most servers limit the number of simultaneous connections that can originate from a single IP address. The application should not make any assumptions about the sequence in which background transfers are performed or the order in which they may complete.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.AsyncPutFile Overload List | AsyncGetFile Method

HttpClient.AttachThread Method

Attach an instance of the class to the current thread

[Visual Basic]
```
Public Function AttachThread() As Boolean
```

[C#]
```
public bool AttachThread();
```

Return Value

A boolean value which specifies if the client could be attached to the current thread. If this method returns **false**, the client could not be attached to the thread and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

When an instance of the class is created it is associated with the current thread that created it. Normally, if another thread attempts to perform an operation using that instance, an error is returned. This is used to ensure that other threads cannot interfere with an operation being performed by the owner thread. In some cases, it may be desirable for one thread in an application to create an instance of the class, establish a connection and then pass that instance to another worker thread. The **AttachThread** method can be used to change the ownership of the class instance to the new worker thread.

This method should be called by the new thread immediately after it has been created, and if the new thread does not release the handle itself, the ownership of the handle should be restored by the original thread. Under no circumstances should **AttachThread** be used to forcibly destroy an instance of a class allocated by another thread while a blocking operation is in progress. To cancel a blocking operation, use the **Cancel** method and then delete the class instance after the blocking function exits and control is returned to the current thread.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.Authenticate Method

Authenticate the client session with a username and password.

Overload List

Authenticate the client session with a username and password.

public bool Authenticate(string,string);

Authenticate the client session with a username and password.

public bool Authenticate(string,string,HttpAuthentication);

See Also

HttpClient Class | SocketTools Namespace | Authentication Property | BearerToken Property | Password Property | UserName Property | SetHeader Method

HttpClient.Authenticate Method (String, String)

Authenticate the client session with a username and password.

[Visual Basic]

```
Overloads Public Function Authenticate( _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]

```
public bool Authenticate(
   string userName,
   string userPassword
);
```

Parameters

userName

A string which specifies the username used to authenticate the client session.

userPassword

A string which specifies the password which will be used to authenticate the client session with the remote host. Not all server resources require the client to authenticate the session. If you are using OAuth 2.0 authentication, this parameter specifies the bearer token.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method will set the Authorization request header for the client session using the credentials provided by the caller. It will always override any custom Authorization header value which may have been previously set using the **SetHeader** method.

If both the **userName** and **userPassword** parameters specify empty strings, the current authentication type will always be set to **HttpAuthentication.authNone**, effectively clearing the current user credentials for the client session.

If you provide a username and password, and the **Authentication** property has not been explicitly set, it will automatically default to using the **HttpAuthentication.authBasic** authentication type.

If you provide a user name and password to the **Connect** method, or you set the **UserName** property and either the **Password** or **BearerToken** property prior to calling the **Connect** method, authentication will be automatically attempted at the time the connection is made. This method is only required if you do not provide user credentials when the connection is established and wish to authenticate the client session at a later time.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Authenticate Overload List | Authentication Property | BearerToken Property | Password Property | UserName Property | SetHeader Method

HttpClient.Authenticate Method (String, String, HttpAuthentication)

Authenticate the client session with a username and password.

[Visual Basic]

```
Overloads Public Function Authenticate( _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal authType As HttpAuthentication _
) As Boolean
```

[C#]

```
public bool Authenticate(
   string userName,
   string userPassword,
   HttpAuthentication authType
);
```

Parameters

userName
A string which specifies the username used to authenticate the client session.

userPassword
A string which specifies the password which will be used to authenticate the client session with the remote host. Not all server resources require the client to authenticate the session. If you are using OAuth 2.0 authentication, this parameter specifies the bearer token.

authType
A HttpAuthentication enumeration value which specifies the authentication method.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method will set the Authorization request header for the client session using the credentials provided by the caller. This method will always override any custom Authorization header value which may have been previously set using the **SetHeader** method.

If both the **userName** and **userPassword** parameters specify empty strings, the current authentication type will always be set to **HttpAuthentication.authNone** regardless of the value of the **authType** parameter. This effectively clears the current user credentials for the client session.

If you provide a user name and password to the **Connect** method, or you set the **UserName** property and either the **Password** or **BearerToken** property prior to calling the **Connect** method, authentication will be automatically attempted at the time the connection is made. This method is only required if you do not provde user credentials when the connection is established and wish to authenticate the client session at a later time.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Authenticate Overload List | Authentication Property | BearerToken Property | Password Property | UserName Property | SetHeader Method

HttpClient.Cancel Method

Cancel the current blocking client operation.

[Visual Basic]

```
Public Sub Cancel()
```

[C#]

```
public void Cancel();
```

Remarks

When the **Cancel** method is called, the blocking client operation will not immediately fail. An internal flag is set which causes the blocking operation to exit with an error. This means that the application cannot cancel an operation and immediately perform some other blocking function. Instead it must allow the calling stack to unwind, returning back to the blocking operation before making any further function calls.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.ClearForm Method

Remove all defined fields from the current form.

[Visual Basic]
```
Public Sub ClearForm()
```

[C#]
```
public void ClearForm();
```

Remarks

The **ClearForm** deletes all form fields, releasing the memory allocated for the field data and resetting the internal state of the current form.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.ClearHeaders Method

Clears the current request and response headers.

[Visual Basic]

```
Public Sub ClearHeaders()
```

[C#]

```
public void ClearHeaders();
```

Remarks

The **ClearHeaders** method clears the request and response headers for the current session, including any cookies which may have been set. This method can be useful in persistent connections, where the client wishes to clear any previously set header values without disconnecting from the server.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.CloseFile Method

Close the file that was opened on the server.

[Visual Basic]

```
Public Function CloseFile() As Boolean
```

[C#]

```
public bool CloseFile();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CloseFile** method is used to close a file that was opened using the **OpenFile** method, or created using the **CreateFile** method. It should be called before the client disconnects from the server.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.Command Method

Send a custom command to the web server.

Overload List

Send a custom command to the web server.

public bool Command(string,string);

Send a custom command to the web server.

public bool Command(string,string,byte[],int);

Send a custom command to the web server.

public bool Command(string,string,string);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.Command Method (String, String)

Send a custom command to the web server.

[Visual Basic]

```
Overloads Public Function Command( _
   ByVal command As String, _
   ByVal resource As String _
) As Boolean
```

[C#]

```
public bool Command(
   string command,
   string resource
);
```

Parameters

command

A string which specifies the command to send.

resource

A string which specifies the resource that the command will be performed upon. The resource may be a file, such as an HTML document or an image, or it may be a script used to process data submitted by the client. Note that this argument specifies the name of the resource only, not a complete URL. In most cases, the resource name should be specified using an absolute path (a path that begins with a leading slash character).

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

There are a number of standard commands which may be used, and there are extended commands which depend on the type of server that the client is connected to. Consult the protocol standard and/or the technical reference documentation for the server to determine what commands may be issued by a client application. An example of some common HTTP commands are:

Command	Description
GET	Return the contents of the specified resource. This command is recognized by all servers.
HEAD	Return only header information for the specified resource. This command is recognized by servers that support at least version 1.0 of the protocol.
POST	Post data to the specified resource. This command is recognized by servers that support at least version 1.0 of the protocol.
PUT	Create or replace the specified resource on the server. This command is recognized by servers that support at least version 1.0 of the protocol. Not all servers support this command.

DELETE	Delete the specified resource from the server. This command is recognized by servers that support at least version 1.1 of the protocol. Not all servers support this command.

Not all servers support all of the listed commands, and some commands may require specific changes to the server configuration. In particular, the PUT and DELETE commands typically require that configuration changes be made by the site administrator. All servers will support the use of the GET command, and all servers that support at least version 1.0 of the protocol will support the POST command.

Only one request may be in progress at one time for each client session. Use the **CloseFile** method to terminate the request after all of the data has been read from the server.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Command Overload List

HttpClient.Command Method (String, String, Byte[], Int32)

Send a custom command to the web server.

```
[Visual Basic]
Overloads Public Function Command( _
   ByVal command As String, _
   ByVal resource As String, _
   ByVal parameter As Byte(), _
   ByVal length As Integer _
) As Boolean
```

```
[C#]
public bool Command(
   string command,
   string resource,
   byte[] parameter,
   int length
);
```

Parameters

command

A string which specifies the command to send.

resource

A string which specifies the resource that the command will be performed upon. The resource may be a file, such as an HTML document or an image, or it may be a script used to process data submitted by the client. Note that this argument specifies the name of the resource only, not a complete URL. In most cases, the resource name should be specified using an absolute path (a path that begins with a leading slash character).

parameter

A byte array which specifies one or more parameters to be sent along with the command. The parameter data is encoded according to the encoding type specified by the **Encoding** property. If the resource does not require any parameters, this argument should be omitted.

length

An integer value which specifies the length of the parameter data in bytes. This value cannot be larger than the size of the byte array passed to this method.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

There are a number of standard commands which may be used, and there are extended commands which depend on the type of server that the client is connected to. Consult the protocol standard and/or the technical reference documentation for the server to determine what commands may be issued by a client application. An example of some common HTTP commands are:

Command	Description
GET	Return the contents of the specified resource. This command is recognized by all servers.

HEAD	Return only header information for the specified resource. This command is recognized by servers that support at least version 1.0 of the protocol.
POST	Post data to the specified resource. This command is recognized by servers that support at least version 1.0 of the protocol.
PUT	Create or replace the specified resource on the server. This command is recognized by servers that support at least version 1.0 of the protocol. Not all servers support this command.
DELETE	Delete the specified resource from the server. This command is recognized by servers that support at least version 1.1 of the protocol. Not all servers support this command.

Not all servers support all of the listed commands, and some commands may require specific changes to the server configuration. In particular, the PUT and DELETE commands typically require that configuration changes be made by the site administrator. All servers will support the use of the GET command, and all servers that support at least version 1.0 of the protocol will support the POST command.

The ***parameter*** argument is used to pass additional information to the server when a resource is requested. This is most commonly used to provide information to scripts, similar to how arguments are used when executing a program from the command line. Unless the POST command is being executed, the data in the buffer will automatically be encoded using the current encoding mechanism specified for the client.

By default, the parameter data is URL encoded, which means that any spaces and non-printable characters are converted to printable characters before submitted to the server. The type of encoding that is performed can be changed by setting the **Encoding** property. Although the default encoding is appropriate for most applications, those that submit XML formatted data may need to change the encoding type.

Only one request may be in progress at one time for each client session. Use the **CloseFile** method to terminate the request after all of the data has been read from the server.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Command Overload List

HttpClient.Command Method (String, String, String)

Send a custom command to the web server.

[Visual Basic]

```
Overloads Public Function Command( _
   ByVal command As String, _
   ByVal resource As String, _
   ByVal parameter As String _
) As Boolean
```

[C#]

```
public bool Command(
   string command,
   string resource,
   string parameter
);
```

Parameters

command

A string which specifies the command to send.

resource

A string which specifies the resource that the command will be performed upon. The resource may be a file, such as an HTML document or an image, or it may be a script used to process data submitted by the client. Note that this argument specifies the name of the resource only, not a complete URL. In most cases, the resource name should be specified using an absolute path (a path that begins with a leading slash character).

parameter

A string specifies one or more parameters to be sent along with the command. The parameter data is encoded according to the encoding type specified by the **Encoding** property. If the resource does not require any parameters, this argument should be omitted.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

There are a number of standard commands which may be used, and there are extended commands which depend on the type of server that the client is connected to. Consult the protocol standard and/or the technical reference documentation for the server to determine what commands may be issued by a client application. An example of some common HTTP commands are:

Command	Description
GET	Return the contents of the specified resource. This command is recognized by all servers.
HEAD	Return only header information for the specified resource. This command is recognized by servers that support at least version 1.0 of the protocol.
POST	Post data to the specified resource. This command is recognized by servers that support at least

	version 1.0 of the protocol.
PUT	Create or replace the specified resource on the server. This command is recognized by servers that support at least version 1.0 of the protocol. Not all servers support this command.
DELETE	Delete the specified resource from the server. This command is recognized by servers that support at least version 1.1 of the protocol. Not all servers support this command.

Not all servers support all of the listed commands, and some commands may require specific changes to the server configuration. In particular, the PUT and DELETE commands typically require that configuration changes be made by the site administrator. All servers will support the use of the GET command, and all servers that support at least version 1.0 of the protocol will support the POST command.

The ***parameter*** argument is used to pass additional information to the server when a resource is requested. This is most commonly used to provide information to scripts, similar to how arguments are used when executing a program from the command line. Unless the POST command is being executed, the data in the buffer will automatically be encoded using the current encoding mechanism specified for the client.

By default, the parameter data is URL encoded, which means that any spaces and non-printable characters are converted to printable characters before submitted to the server. The type of encoding that is performed can be changed by setting the **Encoding** property. Although the default encoding is appropriate for most applications, those that submit XML formatted data may need to change the encoding type.

Only one request may be in progress at one time for each client session. Use the **CloseFile** method to terminate the request after all of the data has been read from the server.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Command Overload List

HttpClient.Connect Method

Establish a connection with a remote host.

Overload List

Establish a connection with a remote host.

public bool Connect();

Establish a connection with a remote host.

public bool Connect(string);

Establish a connection with a remote host.

public bool Connect(string,int);

Establish a connection with a remote host.

public bool Connect(string,int,int);

Establish a connection with a remote host.

public bool Connect(string,int,int,HttpOptions);

Establish a connection with a remote host.

public bool Connect(string,int,int,HttpOptions,HttpVersion);

Establish a connection with a remote host.

public bool Connect(string,int,string,string);

Establish a connection with a remote host.

public bool Connect(string,int,string,string,int);

Establish a connection with a remote host.

public bool Connect(string,int,string,string,int,HttpOptions,HttpVersion);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.Connect Method (String, Int32)

Establish a connection with a remote host.

[Visual Basic]
```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer _
) As Boolean
```

[C#]
```
public bool Connect(
   string hostName,
   int hostPort
);
```

Parameters

hostName
A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort
An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection using assigned property values.

The value of the **UserName** property will be used to determine the username if authentication is required.

The value of the **Password** property will be used to determine the username if authentication is required.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

The value of the **ProtocolVersion** property will be used to specify the protocol version.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Connect Overload List

HttpClient.Connect Method (String, Int32, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   int timeout
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection using assigned property values.

The value of the **UserName** property will be used to determine the username if authentication is required.

The value of the **Password** property will be used to determine the username if authentication is required.

The value of the **Options** property will be used to specify the default options for the connection.

The value of the **ProtocolVersion** property will be used to specify the protocol version.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Connect Overload List

HttpClient.Connect Method (String, Int32, Int32, HttpOptions)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer, _
   ByVal options As HttpOptions _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   int timeout,
   HttpOptions options
);
```

Parameters

hostName
: A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort
: An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

timeout
: An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

options
: One or more of the HttpOptions enumeration flags.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection using assigned property values.

The value of the **UserName** property will be used to determine the username if authentication is required.

The value of the **Password** property will be used to determine the username if authentication is required.

The value of the **Timeout** property will be used to specify the timeout period.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Connect Overload List

HttpClient.Connect Method (String, Int32, String, String)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword
);
```

Parameters

hostName
A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort
An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName
A string which specifies a username used to authenticate the client session. This argument is only required if access to the resource requires authentication.

userPassword
A string which specifies the password used to authenticate the client session. This argument is only required if access to the resource requires authentication.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection using assigned property values.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

The value of the **ProtocolVersion** property will be used to specify the protocol version.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Connect Overload List

HttpClient.Connect Method (String, Int32, String, String, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   int timeout
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName

A string which specifies a username used to authenticate the client session. This argument is only required if access to the resource requires authentication.

userPassword

A string which specifies the password used to authenticate the client session. This argument is only required if access to the resource requires authentication.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection using assigned property values.

The value of the **Options** property will be used to specify the default options for the connection.

The value of the **ProtocolVersion** property will be used to specify the protocol version.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Connect Overload List

HttpClient.Connect Method (String, Int32, Int32, HttpOptions, HttpVersion)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer, _
   ByVal options As HttpOptions, _
   ByVal version As HttpVersion _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   int timeout,
   HttpOptions options,
   HttpVersion version
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

options

One or more of the HttpOptions enumeration flags.

version

One of the HttpVersion enumeration values.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection using assigned property values.

The value of the **UserName** property will be used to determine the username if authentication is required.

The value of the **Password** property will be used to determine the username if authentication is required.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Connect Overload List

HttpClient.Connect Method (String, Int32, String, String, Int32, HttpOptions, HttpVersion)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal timeout As Integer, _
   ByVal options As HttpOptions, _
   ByVal version As HttpVersion _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   int timeout,
   HttpOptions options,
   HttpVersion version
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName

A string which specifies a username used to authenticate the client session. This argument is only required if access to the resource requires authentication.

userPassword

A string which specifies the password used to authenticate the client session. This argument is only required if access to the resource requires authentication.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

options

One or more of the HttpOptions enumeration flags.

version

One of the HttpVersion enumeration values.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Connect Overload List

HttpClient.CreateFile Method

Create a new file or overwrite an existing file on the web server.

Overload List

Create a new file or overwrite an existing file on the web server.

public bool CreateFile(string);

Create a new file or overwrite an existing file on the web server.

public bool CreateFile(string,int);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.CreateFile Method (String)

Create a new file or overwrite an existing file on the web server.

[Visual Basic]

```
Overloads Public Function CreateFile( _
   ByVal resourceName As String _
) As Boolean
```

[C#]

```
public bool CreateFile(
   string resourceName
);
```

Parameters

resourceName

A string which specifies the name of the file being created on the server. The client must have the appropriate access rights to create the file or an error will be returned.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method uses the PUT command to create the file. The server must support this command and the user must have the appropriate permission to create the specified file. If this method is successful, the client should then use the **Write** method to send the contents of the file to the server. Once all of the data has been written, the **CloseFile** method should be called to close the file and complete the operation. Note that this method is typically only accepted by servers that support version 1.1 of the protocol or later.

When using **Write** to send the contents of the file to the server, it is recommended that the data be written in logical blocks that are no larger than 8,192 bytes in size. Attempting to write very large amounts of data in a single call can either cause the current thread to block or, in the case of an asynchronous connection, return an error if the internal buffers cannot accommodate all of the data. To send the entire contents of a file at once, use the **PutData** method instead of calling **CreateFile**.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.CreateFile Overload List

HttpClient.CreateFile Method (String, Int32)

Create a new file or overwrite an existing file on the web server.

[Visual Basic]

```
Overloads Public Function CreateFile( _
   ByVal resourceName As String, _
   ByVal fileSize As Integer _
) As Boolean
```

[C#]

```
public bool CreateFile(
   string resourceName,
   int fileSize
);
```

Parameters

resourceName
: A string which specifies the name of the file being created on the server. The client must have the appropriate access rights to create the file or an error will be returned.

fileSize
: A number which specifies the size of the file in bytes. This value must be greater than zero.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method uses the PUT command to create the file. The server must support this command and the user must have the appropriate permission to create the specified file. If this method is successful, the client should then use the **Write** method to send the contents of the file to the server. Once all of the data has been written, the **CloseFile** method should be called to close the file and complete the operation. Note that this method is typically only accepted by servers that support version 1.1 of the protocol or later.

When using **Write** to send the contents of the file to the server, it is recommended that the data be written in logical blocks that are no larger than 8,192 bytes in size. Attempting to write very large amounts of data in a single call can either cause the current thread to block or, in the case of an asynchronous connection, return an error if the internal buffers cannot accommodate all of the data. To send the entire contents of a file at once, use the **PutData** method instead of calling **CreateFile**.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.CreateFile Overload List

HttpClient.CreateForm Method

Create a new virtual form.

Overload List

Create a new virtual form.

public bool CreateForm();

Create a new virtual form.

public bool CreateForm(string);

Create a new virtual form.

public bool CreateForm(string,HttpFormMethod);

Create a new virtual form.

public bool CreateForm(string,HttpFormMethod,HttpFormType);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.CreateForm Method ()

Create a new virtual form.

[Visual Basic]

```
Overloads Public Function CreateForm() As Boolean
```

[C#]

```
public bool CreateForm();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateForm** method is used to create a new form that will be populated with values and then submitted to the server for processing. Any previously defined form data will be deleted when this method is called.

This method creates the form using assigned property values.

The value of the **FormAction** property will be used to determine the script that will accept the form data on the server.

The value of the **FormMethod** property will be used to determine the method used to submit the form data.

The value of the **FormType** property will be used to specify the form type and encoding method.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.CreateForm Overload List

HttpClient.CreateForm Method (String)

Create a new virtual form.

[Visual Basic]

```
Overloads Public Function CreateForm( _
   ByVal formAction As String _
) As Boolean
```

[C#]

```
public bool CreateForm(
   string formAction
);
```

Parameters

formAction

A string which specifies the name of the resource that the form data will be submitted to. Typically this is the name of a script that is executed on the server. If this argument is omitted, the value of the **FormAction** property is used as the default value. If the **FormAction** property is undefined, the value of the **Resource** property will be used as the default value.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateForm** method is used to create a new form that will be populated with values and then submitted to the server for processing. Any previously defined form data will be deleted when this method is called.

This method creates the form using assigned property values.

The value of the **FormMethod** property will be used to determine the method used to submit the form data.

The value of the **FormType** property will be used to specify the form type and encoding method.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.CreateForm Overload List

HttpClient.CreateForm Method (String, HttpFormMethod)

Create a new virtual form.

[Visual Basic]

```
Overloads Public Function CreateForm( _
   ByVal formAction As String, _
   ByVal formMethod As HttpFormMethod _
) As Boolean
```

[C#]

```
public bool CreateForm(
   string formAction,
   HttpFormMethod formMethod
);
```

Parameters

formAction

A string which specifies the name of the resource that the form data will be submitted to. Typically this is the name of a script that is executed on the server. If this argument is omitted, the value of the **FormAction** property is used as the default value. If the **FormAction** property is undefined, the value of the **Resource** property will be used as the default value.

formMethod

An HttpFormMethod enumeration value which specifies how the form data will be submitted to the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateForm** method is used to create a new form that will be populated with values and then submitted to the server for processing. Any previously defined form data will be deleted when this method is called.

This method creates the form using assigned property values.

The value of the **FormType** property will be used to specify the form type and encoding method.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.CreateForm Overload List

HttpClient.CreateForm Method (String, HttpFormMethod, HttpFormType)

Create a new virtual form.

[Visual Basic]

```
Overloads Public Function CreateForm( _
   ByVal formAction As String, _
   ByVal formMethod As HttpFormMethod, _
   ByVal formType As HttpFormType _
) As Boolean
```

[C#]

```
public bool CreateForm(
   string formAction,
   HttpFormMethod formMethod,
   HttpFormType formType
);
```

Parameters

formAction
A string which specifies the name of the resource that the form data will be submitted to. Typically this is the name of a script that is executed on the server. If this argument is omitted, the value of the **FormAction** property is used as the default value. If the **FormAction** property is undefined, the value of the **Resource** property will be used as the default value.

formMethod
An HttpFormMethod enumeration value which specifies how the form data will be submitted to the server.

formType
An HttpFormType enumeration value which specifies the type of form and how the form data will be encoded.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateForm** method is used to create a new form that will be populated with values and then submitted to the server for processing. Any previously defined form data will be deleted when this method is called.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.CreateForm Overload List

HttpClient.DeleteField Method

Delete a form field and its value from the current form.

[Visual Basic]

```
Public Function DeleteField( _
   ByVal fieldName As String _
) As Boolean
```

[C#]

```
public bool DeleteField(
   string fieldName
);
```

Parameters

fieldName

A string which specifies the name of the field to remove from the form.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **DeleteField** method is used to remove a field and its associated value from the current form. The memory allocated for the field data will be released.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.DeleteFile Method

Remove a file on the web server.

[Visual Basic]

```
Public Function DeleteFile( _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool DeleteFile(
   string fileName
);
```

Parameters

fileName

A string which specifies the name of the file to delete.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **DeleteFile** method deletes an existing file or resource from the web server using the DELETE command. This command is typically only accepted by servers that support version 1.1 of the protocol or later. This method requires the user have the appropriate permissions to remove the file or resource.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.Disconnect Method

Terminate the connection with a remote host.

[Visual Basic]

```
Public Sub Disconnect()
```

[C#]

```
public void Disconnect();
```

Remarks

The **Disconnect** method terminates the connection with the remote host and releases the client handle allocated by the class. Note that the socket is not immediately released when the connection is terminated and will enter a wait state for two minutes. After the time wait period has elapsed, the client will be released by the operating system. This is a normal safety mechanism to handle any packets that may arrive after the connection has been closed.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.Dispose Method

Releases all resources used by HttpClient.

Overload List

Releases all resources used by HttpClient.

public void Dispose();

Releases the unmanaged resources allocated by the HttpClient class and optionally releases the managed resources.

protected virtual void Dispose(bool);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.Dispose Method ()

Releases all resources used by HttpClient.

[Visual Basic]

```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]

```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Dispose Overload List

HttpClient.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the HttpClient class and optionally releases the managed resources.

```
[Visual Basic]
Overridable Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

```
[C#]
protected virtual void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **HttpClient** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Dispose Overload List

HttpClient.Finalize Method

Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.

[Visual Basic]

```
Overrides Protected Sub Finalize()
```

[C#]

```
protected override void Finalize();
```

See Also

HttpClient Class | SocketTools Namespace

HttpClient.GetCookie Method

Return information about the specified cookie.

Overload List

Return information about the specified cookie.

public bool GetCookie(string,ref HttpCookie);

Return information about the specified cookie.

public bool GetCookie(string,ref string);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.GetCookie Method (String, HttpCookie)

Return information about the specified cookie.

```
[Visual Basic]
Overloads Public Function GetCookie( _
   ByVal cookieName As String, _
   ByRef cookieInfo As HttpCookie _
) As Boolean
```

```
[C#]
public bool GetCookie(
   string cookieName,
   ref HttpCookie cookieInfo
);
```

Parameters

cookieName
A string which specifies the name of the cookie to return information about. To obtain a list of cookies which have been set by the server, use the **CookieCount** and **CookieName** properties.

cookieInfo
An HttpCookie structure which will contain information about the cookie when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The Hypertext Transfer Protocol uses special tokens called "cookies" to maintain persistent state information between requests for a resource. These cookies are exchanged between the client and server by setting specific header fields. When a server wants the client to use a cookie, it will include a header field named Set-Cookie in the response header when the client requests a resource. The client can then take this cookie and store it, either temporarily in memory or permanently in a file on the local system. The next time that the client requests a resource from that server, it can send the cookie back to the server by setting the Cookie header field. The **GetCookie** method searches for a cookie set by the server in the Set-Cookie header field. The **SetCookie** method creates or modifies the Cookie header field for the next resource requested by the client.

There are two general types of cookies that are used by servers. Session cookies exist only for the duration of the client session; they are stored in memory and not saved in any kind of permanent storage. When the client application terminates, session cookies are deleted and no longer used. Persistent cookies are stored on the local system and are used by the client until their expiration time. It is the responsibility of the client application to store persistent cookies; applications may use a flat text file, a database or any other storage method available.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.GetCookie Overload List

HttpClient.GetCookie Method (String, String)

Return information about the specified cookie.

[Visual Basic]

```
Overloads Public Function GetCookie( _
   ByVal cookieName As String, _
   ByRef cookieValue As String _
) As Boolean
```

[C#]

```
public bool GetCookie(
   string cookieName,
   ref string cookieValue
);
```

Parameters

cookieName

A string which specifies the name of the cookie to return information about. To obtain a list of cookies which have been set by the server, use the **CookieCount** and **CookieName** properties.

cookieValue

A string passed by reference which will contain the value of the specified cookie when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The Hypertext Transfer Protocol uses special tokens called "cookies" to maintain persistent state information between requests for a resource. These cookies are exchanged between the client and server by setting specific header fields. When a server wants the client to use a cookie, it will include a header field named Set-Cookie in the response header when the client requests a resource. The client can then take this cookie and store it, either temporarily in memory or permanently in a file on the local system. The next time that the client requests a resource from that server, it can send the cookie back to the server by setting the Cookie header field. The **GetCookie** method searches for a cookie set by the server in the Set-Cookie header field. The **SetCookie** method creates or modifies the Cookie header field for the next resource requested by the client.

There are two general types of cookies that are used by servers. Session cookies exist only for the duration of the client session; they are stored in memory and not saved in any kind of permanent storage. When the client application terminates, session cookies are deleted and no longer used. Persistent cookies are stored on the local system and are used by the client until their expiration time. It is the responsibility of the client application to store persistent cookies; applications may use a flat text file, a database or any other storage method available.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.GetCookie Overload List

HttpClient.GetData Method

Transfer data from the web server and store it in a local buffer.

Overload List

Transfer data from the web server and store it in a local buffer.

public bool GetData(byte[],ref int);

Transfer data from the web server and store it in a local buffer.

public bool GetData(string,byte[],ref int);

Transfer data from the web server and store it in a local buffer.

public bool GetData(string,byte[],ref int,HttpTransferOptions);

Transfers the contents of a file from the server and stores it in a MemoryStream.

public bool GetData(string,MemoryStream);

Transfers the contents of a file from the server and stores it in a MemoryStream.

public bool GetData(string,MemoryStream,HttpTransferOptions);

Transfer data from the web server and store it in a local buffer.

public bool GetData(string,ref string);

Transfer data from the web server and store it in a local buffer.

public bool GetData(string,ref string,HttpTransferOptions);

Transfer data from the web server and store it in a local buffer.

public bool GetData(ref string);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.GetData Method (String)

Transfer data from the web server and store it in a local buffer.

[Visual Basic]

```
Overloads Public Function GetData( _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool GetData(
   ref string buffer
);
```

Parameters

buffer
: A string passed by reference that the data will be stored in.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetData** method transfers data from the server to the local system, storing it in the specified buffer. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.GetData Overload List

HttpClient.GetData Method (String, String)

Transfer data from the web server and store it in a local buffer.

[Visual Basic]

```
Overloads Public Function GetData( _
   ByVal resourceName As String, _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool GetData(
   string resourceName,
   ref string buffer
);
```

Parameters

resourceName

A string that specifies the resource on the server that will be accessed. If the resource specifies a file, then the contents of the file will be returned by the server. If the resource specifies a script or other executable content, it will be executed and the output will be transferred to the local system. The resource name should be specified using an absolute path that begins with a leading slash character.

buffer

A string passed by reference that the data will be stored in.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetData** method transfers data from the server to the local system, storing it in the specified buffer. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.GetData Overload List

HttpClient.GetData Method (String, String, HttpTransferOptions)

Transfer data from the web server and store it in a local buffer.

[Visual Basic]

```
Overloads Public Function GetData( _
   ByVal resourceName As String, _
   ByRef buffer As String, _
   ByVal options As HttpTransferOptions _
) As Boolean
```

[C#]

```
public bool GetData(
   string resourceName,
   ref string buffer,
   HttpTransferOptions options
);
```

Parameters

resourceName

A string that specifies the resource on the server that will be accessed. If the resource specifies a file, then the contents of the file will be returned by the server. If the resource specifies a script or other executable content, it will be executed and the output will be transferred to the local system. The resource name should be specified using an absolute path that begins with a leading slash character.

buffer

A string passed by reference that the data will be stored in.

options

An HttpTransferOptions enumeration value which specifies one or more options when transferring the data from the server to the local system.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetData** method transfers data from the server to the local system, storing it in the specified buffer. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.GetData Overload List

HttpClient.GetData Method (Byte[], Int32)

Transfer data from the web server and store it in a local buffer.

[Visual Basic]
```
Overloads Public Function GetData( _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]
```
public bool GetData(
   byte[] buffer,
   ref int length
);
```

Parameters

buffer
A byte array that the data will be stored in.

length
An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetData** method transfers data from the server to the local system, storing it in the specified buffer. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.GetData Overload List

HttpClient.GetData Method (String, Byte[], Int32, HttpTransferOptions)

Transfer data from the web server and store it in a local buffer.

[Visual Basic]

```
Overloads Public Function GetData( _
   ByVal resourceName As String, _
   ByVal buffer As Byte(), _
   ByRef length As Integer, _
   ByVal options As HttpTransferOptions _
) As Boolean
```

[C#]

```
public bool GetData(
   string resourceName,
   byte[] buffer,
   ref int length,
   HttpTransferOptions options
);
```

Parameters

resourceName
A string that specifies the resource on the server that will be accessed. If the resource specifies a file, then the contents of the file will be returned by the server. If the resource specifies a script or other executable content, it will be executed and the output will be transferred to the local system. The resource name should be specified using an absolute path that begins with a leading slash character.

buffer
A byte array that the data will be stored in.

length
An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

options
An HttpTransferOptions enumeration value which specifies one or more options when transferring the data from the server to the local system.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetData** method transfers data from the server to the local system, storing it in the specified buffer. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.GetData Overload List

HttpClient.GetData Method (String, Byte[], Int32)

Transfer data from the web server and store it in a local buffer.

[Visual Basic]

```
Overloads Public Function GetData( _
   ByVal resourceName As String, _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool GetData(
   string resourceName,
   byte[] buffer,
   ref int length
);
```

Parameters

resourceName

A string that specifies the resource on the server that will be accessed. If the resource specifies a file, then the contents of the file will be returned by the server. If the resource specifies a script or other executable content, it will be executed and the output will be transferred to the local system. The resource name should be specified using an absolute path that begins with a leading slash character.

buffer

A byte array that the data will be stored in.

length

An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetData** method transfers data from the server to the local system, storing it in the specified buffer. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.GetData Overload List

HttpClient.GetData Method (String, MemoryStream, HttpTransferOptions)

Transfers the contents of a file from the server and stores it in a MemoryStream.

[Visual Basic]

```
Overloads Public Function GetData( _
   ByVal resourceName As String, _
   ByVal memStream As MemoryStream, _
   ByVal options As HttpTransferOptions _
) As Boolean
```

[C#]

```
public bool GetData(
   string resourceName,
   MemoryStream memStream,
   HttpTransferOptions options
);
```

Parameters

resourceName

A string that specifies the resource on the server that will be accessed. If the resource specifies a file, then the contents of the file will be returned by the server. If the resource specifies a script or other executable content, it will be executed and the output will be transferred to the local system. The resource name should be specified using an absolute path that begins with a leading slash character.

memStream

A System.IO.MemoryStream object that will contain the file data when the method returns. This stream must be open and writable.

options

An HttpTransferOptions enumeration value which specifies one or more options when transferring the data from the server to the local system.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetData** method transfers data from a file on the server to the local system, storing it in the specified MemoryStream. This method will cause the calling current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The contents of the MemoryStream will be replaced by the contents of the file and the current position will be reset to the beginning of the stream. The stream must be open and writable, otherwise this method will throw **System.NotSupportedException**.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.GetData Overload List

HttpClient.GetData Method (String, MemoryStream)

Transfers the contents of a file from the server and stores it in a MemoryStream.

[Visual Basic]

```
Overloads Public Function GetData( _
   ByVal resourceName As String, _
   ByVal memStream As MemoryStream _
) As Boolean
```

[C#]

```
public bool GetData(
   string resourceName,
   MemoryStream memStream
);
```

Parameters

resourceName

A string that specifies the resource on the server that will be accessed. If the resource specifies a file, then the contents of the file will be returned by the server. If the resource specifies a script or other executable content, it will be executed and the output will be transferred to the local system. The resource name should be specified using an absolute path that begins with a leading slash character.

memStream

A System.IO.MemoryStream object that will contain the file data when the method returns. This stream must be open and writable.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetData** method transfers data from a file on the server to the local system, storing it in the specified MemoryStream. This method will cause the calling current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The contents of the MemoryStream will be replaced by the contents of the file and the current position will be reset to the beginning of the stream. The stream must be open and writable, otherwise this method will throw **System.NotSupportedException**.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.GetData Overload List

HttpClient.GetFile Method

Transfer data from the web server and store it in a file on the local system.

Overload List

Transfer data from the web server and store it in a file on the local system.

public bool GetFile(string);

Transfer data from the web server and store it in a file on the local system.

public bool GetFile(string,HttpTransferOptions);

Transfer data from the web server and store it in a file on the local system.

public bool GetFile(string,string);

Transfer data from the web server and store it in a file on the local system.

public bool GetFile(string,string,HttpTransferOptions);

Transfer data from the web server and store it in a file on the local system.

public bool GetFile(string,string,HttpTransferOptions,long);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.GetFile Method (String)

Transfer data from the web server and store it in a file on the local system.

[Visual Basic]

```
Overloads Public Function GetFile( _
   ByVal localFile As String _
) As Boolean
```

[C#]

```
public bool GetFile(
   string localFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be created. If the file already exists, it will be overwritten. The file pathing and name conventions must be that of the local host.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFile** method downloads a file from the server to the local system. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.GetFile Overload List

HttpClient.GetFile Method (String, String)

Transfer data from the web server and store it in a file on the local system.

[Visual Basic]

```
Overloads Public Function GetFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String _
) As Boolean
```

[C#]

```
public bool GetFile(
   string localFile,
   string remoteFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be created. If the file already exists, it will be overwritten. The file pathing and name conventions must be that of the local host.

remoteFile

A string that specifies the resource on the server that will be accessed. If the resource specifies a file, then the contents of the file will be returned by the server. If the resource specifies a script or other executable content, it will be executed and the output will be transferred to the local system. The resource name should be specified using an absolute path that begins with a leading slash character.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFile** method downloads a file from the server to the local system. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.GetFile Overload List

HttpClient.GetFile Method (String, HttpTransferOptions)

Transfer data from the web server and store it in a file on the local system.

[Visual Basic]

```
Overloads Public Function GetFile( _
   ByVal localFile As String, _
   ByVal options As HttpTransferOptions _
) As Boolean
```

[C#]

```
public bool GetFile(
   string localFile,
   HttpTransferOptions options
);
```

Parameters

localFile

A string that specifies the file on the local system that will be created. If the file already exists, it will be overwritten. The file pathing and name conventions must be that of the local host.

options

An HttpTransferOptions enumeration value which specifies one or more options when transferring the data from the server to the local system.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFile** method downloads a file from the server to the local system. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.GetFile Overload List

HttpClient.GetFile Method (String, String, HttpTransferOptions)

Transfer data from the web server and store it in a file on the local system.

[Visual Basic]

```
Overloads Public Function GetFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal options As HttpTransferOptions _
) As Boolean
```

[C#]

```
public bool GetFile(
   string localFile,
   string remoteFile,
   HttpTransferOptions options
);
```

Parameters

localFile

A string that specifies the file on the local system that will be created. If the file already exists, it will be overwritten. The file pathing and name conventions must be that of the local host.

remoteFile

A string that specifies the resource on the server that will be accessed. If the resource specifies a file, then the contents of the file will be returned by the server. If the resource specifies a script or other executable content, it will be executed and the output will be transferred to the local system. The resource name should be specified using an absolute path that begins with a leading slash character.

options

An HttpTransferOptions enumeration value which specifies one or more options when transferring the data from the server to the local system.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFile** method downloads a file from the server to the local system. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.GetFile Overload List

HttpClient.GetFile Method (String, String, HttpTransferOptions, Int64)

Transfer data from the web server and store it in a file on the local system.

[Visual Basic]

```
Overloads Public Function GetFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal options As HttpTransferOptions, _
   ByVal offset As Long _
) As Boolean
```

[C#]

```
public bool GetFile(
   string localFile,
   string remoteFile,
   HttpTransferOptions options,
   long offset
);
```

Parameters

localFile
: A string that specifies the file on the local system that will be created. If the file already exists, it will be overwritten. The file pathing and name conventions must be that of the local host.

remoteFile
: A string that specifies the resource on the server that will be accessed. If the resource specifies a file, then the contents of the file will be returned by the server. If the resource specifies a script or other executable content, it will be executed and the output will be transferred to the local system. The resource name should be specified using an absolute path that begins with a leading slash character.

options
: An HttpTransferOptions enumeration value which specifies one or more options when transferring the data from the server to the local system.

offset
: A integer value which specifies the offset where the file transfer should begin. A value of zero specifies that the file transfer should start at the beginning of the file. A value greater than zero is typically used to restart a transfer that has not completed successfully. Note that specifying a non-zero offset requires that the server support the option to restart file transfers.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFile** method downloads a file from the server to the local system. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.GetFile Overload List

HttpClient.GetFileSize Method

Return the size of the specified file on the web server.

Overload List

Return the size of the specified file on the web server.

public bool GetFileSize(string,ref int);

Return the size of the specified file on the web server.

public bool GetFileSize(string,ref long);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.GetFileSize Method (String, Int32)

Return the size of the specified file on the web server.

[Visual Basic]

```
Overloads Public Function GetFileSize( _
   ByVal fileName As String, _
   ByRef fileSize As Integer _
) As Boolean
```

[C#]

```
public bool GetFileSize(
   string fileName,
   ref int fileSize
);
```

Parameters

fileName

A string that specifies a file on the server. The file name should be specified using an absolute path that begins with a leading slash character.

fileSize

An integer passed by reference which will contain the size of the file in bytes when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method uses the HEAD command to retrieve header information about the file without downloading the contents of the file itself. This requires that the server support at least version 1.0 of the protocol standard, or an error will be returned.

The server may not return a file size for some resources. This is typically the case with scripts that generate dynamic content because the server has no way of determining the size of the output generated by the script without actually executing it. The server may also not provide a file size for HTML documents which use server side includes (SSI) because that content is also dynamically created by the server. If the request to the server was successful and the file exists, but the server does not return a file size, the method will succeed but the file size returned to the caller will be zero.

When a request is made to the server for information about the file, the class library will attempt to keep the connection alive, even if the **KeepAlive** property has not been set to **true**. This allows an application to request the file size and then download the file without having to write additional code to re-establish the connection. However, it is possible that the attempt to keep the connection open will fail. In that case, an error will be returned and the session will no longer be valid. If this happens, the method may still return a valid file size. To determine if an error occurred, check the value of the **LastError** property.

Note that if the file on the server is a text file, it is possible that the value returned by this method will not match the size of the file when it is downloaded to the local system. This is because different operating systems use different sequences of characters to mark the end of a line of text, and when a file is transferred in text mode, the end of line character sequence is automatically converted to a carriage return-linefeed, which is the convention used by the Windows platform.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.GetFileSize Overload List

HttpClient.GetFileSize Method (String, Int64)

Return the size of the specified file on the web server.

[Visual Basic]

```
Overloads Public Function GetFileSize( _
   ByVal fileName As String, _
   ByRef fileSize As Long _
) As Boolean
```

[C#]

```
public bool GetFileSize(
   string fileName,
   ref long fileSize
);
```

Parameters

fileName

A string that specifies a file on the server. The file name should be specified using an absolute path that begins with a leading slash character.

fileSize

A long integer passed by reference which will contain the size of the file in bytes when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method uses the HEAD command to retrieve header information about the file without downloading the contents of the file itself. This requires that the server support at least version 1.0 of the protocol standard, or an error will be returned.

The server may not return a file size for some resources. This is typically the case with scripts that generate dynamic content because the server has no way of determining the size of the output generated by the script without actually executing it. The server may also not provide a file size for HTML documents which use server side includes (SSI) because that content is also dynamically created by the server. If the request to the server was successful and the file exists, but the server does not return a file size, the method will succeed but the file size returned to the caller will be zero.

When a request is made to the server for information about the file, the class library will attempt to keep the connection alive, even if the **KeepAlive** property has not been set to **true**. This allows an application to request the file size and then download the file without having to write additional code to re-establish the connection. However, it is possible that the attempt to keep the connection open will fail. In that case, an error will be returned and the session will no longer be valid. If this happens, the method may still return a valid file size. To determine if an error occurred, check the value of the **LastError** property.

Note that if the file on the server is a text file, it is possible that the value returned by this method will not match the size of the file when it is downloaded to the local system. This is because different operating systems use different sequences of characters to mark the end of a line of text, and when a file is transferred in text mode, the end of line character sequence is automatically converted to a carriage return-linefeed, which is the convention used by the Windows platform.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.GetFileSize Overload List

---

HttpClient.GetFileTime Method

Return the modification date and time for specified file on the web server.

Overload List

Return the modification date and time for specified file on the web server.

public bool GetFileTime(string,ref DateTime);

Return the modification date and time for specified file on the web server.

public bool GetFileTime(string,ref DateTime,bool);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.GetFileTime Method (String, DateTime)

Return the modification date and time for specified file on the web server.

[Visual Basic]

```
Overloads Public Function GetFileTime( _
   ByVal fileName As String, _
   ByRef fileDate As Date _
) As Boolean
```

[C#]

```
public bool GetFileTime(
   string fileName,
   ref DateTime fileDate
);
```

Parameters

fileName
: A string that specifies a file on the server. The file name should be specified using an absolute path that begins with a leading slash character.

fileDate
: A **System.DateTime** structure passed by reference which will contain the file modification date and time when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method uses the HEAD command to retrieve header information about the file without downloading the contents of the file itself. This requires that the server support at least version 1.0 of the protocol standard, or an error will be returned.

When a request is made to the server for information about the file, the class library will attempt to keep the connection alive, even if the **KeepAlive** property has not been set to **true**. This allows an application to request the modification time and then download the file without having to write additional code to re-establish the connection. However, it is possible that the attempt to keep the connection open will fail. In that case, an error will be returned and the session will no longer be valid. If this happens, the method may still return a valid date and time. To determine if an error occurred, check the value of the **LastError** property.

The **Localize** property will determine if the returned file time is adjusted for the local timezone.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.GetFileTime Overload List

HttpClient.GetFileTime Method (String, DateTime, Boolean)

Return the modification date and time for specified file on the web server.

[Visual Basic]

```
Overloads Public Function GetFileTime( _
   ByVal fileName As String, _
   ByRef fileDate As Date, _
   ByVal localDate As Boolean _
) As Boolean
```

[C#]

```
public bool GetFileTime(
   string fileName,
   ref DateTime fileDate,
   bool localDate
);
```

Parameters

fileName

A string that specifies a file on the server. The file name should be specified using an absolute path that begins with a leading slash character.

fileDate

A **System.DateTime** structure passed by reference which will contain the file modification date and time when the method returns.

localDate

A boolean value which specifies if the date and time value should be localized for the current timezone.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method uses the HEAD command to retrieve header information about the file without downloading the contents of the file itself. This requires that the server support at least version 1.0 of the protocol standard, or an error will be returned.

When a request is made to the server for information about the file, the class library will attempt to keep the connection alive, even if the **KeepAlive** property has not been set to **true**. This allows an application to request the modification time and then download the file without having to write additional code to re-establish the connection. However, it is possible that the attempt to keep the connection open will fail. In that case, an error will be returned and the session will no longer be valid. If this happens, the method may still return a valid date and time. To determine if an error occurred, check the value of the **LastError** property.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.GetFileTime Overload List

HttpClient.GetFirstHeader Method

Return the first response header field name and value.

[Visual Basic]

```
Public Function GetFirstHeader( _
   ByRef headerField As String, _
   ByRef headerValue As String _
) As Boolean
```

[C#]

```
public bool GetFirstHeader(
   ref string headerField,
   ref string headerValue
);
```

Parameters

headerField

A string passed by reference that will contain the name of the first header field returned by the server.

headerValue

A string passed by reference that will contain the value of the specified header field.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFirstHeader** method is used get the first header field name and value from the response header returned by the server. This method should only be called after the client has requested the resource. This method is typically used in conjunction with the **GetNextHeader** method to enumerate all of the response header fields returned by the server.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.GetHeader Method

Return the value of the specified response header field.

[Visual Basic]

```
Public Function GetHeader( _
   ByVal headerName As String, _
   ByRef headerValue As String _
) As Boolean
```

[C#]

```
public bool GetHeader(
   string headerName,
   ref string headerValue
);
```

Parameters

headerName
A string that specifies the name of the header field.

headerValue
A string passed by reference that will contain the value of the specified header field.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeader** method is used get the value of a specific header field from the response headers returned by the server. This method should only be called after the client has requested the resource. To enumerate all of the header fields returned by the server, use the **GetFirstHeader** and **GetNextHeader** methods.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.GetNextHeader Method

Return the next response header field name and value.

[Visual Basic]

```
Public Function GetNextHeader( _
   ByRef headerField As String, _
   ByRef headerValue As String _
) As Boolean
```

[C#]

```
public bool GetNextHeader(
   ref string headerField,
   ref string headerValue
);
```

Parameters

headerField

A string passed by reference that will contain the name of the first header field returned by the server.

headerValue

A string passed by reference that will contain the value of the specified header field.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetNextHeader** method is used get the next header field name and value from the response header returned by the server. This method should only be called after the client has requested the resource. This method is used in conjunction with the **GetFirstHeader** method to enumerate all of the response header fields returned by the server.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.Initialize Method

Initialize an instance of the HttpClient class.

Overload List

Initialize an instance of the HttpClient class.

public bool Initialize();

Initialize an instance of the HttpClient class.

public bool Initialize(string);

See Also

HttpClient Class | SocketTools Namespace | Uninitialize Method

HttpClient.Initialize Method ()

Initialize an instance of the HttpClient class.

[Visual Basic]
```
Overloads Public Function Initialize() As Boolean
```

[C#]
```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The **Initialize** method can be used to explicitly initialize an instance of the HttpClient class, loading the networking library and allocating resources for the current thread. Typically it is not necessary to explicitly call this method because the instance of the class is initialized by the class constructor. However, if the **Uninitialize** method is called, the class must be re-initialized before any other methods are called.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Initialize Overload List | Uninitialize Method

HttpClient.Initialize Method (String)

Initialize an instance of the HttpClient class.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The **Initialize** method can be used to explicitly initialize an instance of the HttpClient class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the HttpClient class.

Example

The following example shows how to use the Initialize method to initialize an instance of the class. This example assumes that the license key string has been defined in code.

```
SocketTools.HttpClient httpClient = new SocketTools.HttpClient();

if (httpClient.Initialize(strLicenseKey) == false)
{
    MessageBox.Show(httpClient.LastErrorString, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    return;
}
```

```
Dim httpClient As New SocketTools.HttpClient

If httpClient.Initialize(strLicenseKey) = False Then
    MsgBox(httpClient.LastErrorString, vbIconExclamation)
    Exit Sub
End If
```

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

HttpClient.OpenFile Method

Open a file on the web server for reading.

[Visual Basic]

```
Public Function OpenFile( _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool OpenFile(
   string fileName
);
```

Parameters

fileName

A string which specifies the name of the file being opened on the server. The file name should be specified using an absolute path that begins with a leading slash character.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method uses the GET command to retrieve the file. If this method is successful, the client should then use the **Read** method to read the contents of the file from the server. Once all of the data has been read, the **CloseFile** method should be called to close the file and complete the operation. If the file being opened is not an HTML or text document, then it's recommended that you read the data into a byte array.

This method should not be used to post data to a script or other executable resource on the server. If you wish to post data to a script, then the **PostData** method should be used instead.

The client must have the appropriate access rights to open the file for reading or an error will be returned. It may be required that the **UserName** and **Password** properties be set to authenticate the client session so that access to the resource is permitted.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.PatchData Method

Submits patch data to the server and returns the result in a string.

Overload List

Submits patch data to the server and returns the result in a string.

public bool PatchData(string,string,ref string);

Submits patch data to the server and returns the result in a string.

public bool PatchData(string,string,ref string,HttpPatchOptions);

Submits patch data to the server and returns the result in a string.

public bool PatchData(string,ref string);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.PatchData Method (String, String, String, HttpPatchOptions)

Submits patch data to the server and returns the result in a string.

[Visual Basic]

```
Overloads Public Function PatchData( _
   ByVal resourceName As String, _
   ByVal patchData As String, _
   ByRef outputBuffer As String, _
   ByVal options As HttpPatchOptions _
) As Boolean
```

[C#]

```
public bool PatchData(
   string resourceName,
   string patchData,
   ref string outputBuffer,
   HttpPatchOptions options
);
```

Parameters

resourceName
A string that specifies the resource on the server that the data will be posted to. Typically this is the name of an executable script on the server. The resource name should be specified using an absolute path that begins with a leading slash character.

patchData
A string that contains the patch data which will be provided to the server.

outputBuffer
A string passed by reference which will contain the reponse from the server when the method returns.

options
An HttpPatchOptions enumeration that specifies one or more options when posting data to the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PatchData** method is used to submit XML or JSON formatted patch data to a service, and then returns a copy of the response from the server into a string buffer. This method will not perform any encoding and will not automatically define the type of patch data being submitted. Your application is responsible for specifying the content type for the patch data and ensuring that the XML or JSON data being submitted to the server is formatted correctly.

This method sends a PATCH command to the server, which is similar to a POST or PUT request. It is used to make partial updates to a resource, rather than creating or replacing it entirely. The format of the patch data is specific to the service being used. If the resource being patched does not exist, the behavior is defined by the server. If enough information is provided, it may choose to create the resource just as if a PUT command was used, or it may return an error.

Your application should use the **SetHeader** method to define the Content-Type header prior to calling the **PatchData** method. One of the most common formats used is the JSON Merge Patch which is defined in RFC 7396. The value for the Content-Type header for this patch format is "application/merge-patch+json". Refer to your service API documentation to determine what patch formats are acceptable, along with any additional header values which must be defined.

This method will cause the current thread to block until the operation completes, a timeout occurs or the post is canceled. During the operation, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PatchData Overload List

HttpClient.PatchData Method (String, String, String)

Submits patch data to the server and returns the result in a string.

[Visual Basic]

```
Overloads Public Function PatchData( _
   ByVal resourceName As String, _
   ByVal patchData As String, _
   ByRef outputBuffer As String _
) As Boolean
```

[C#]

```
public bool PatchData(
   string resourceName,
   string patchData,
   ref string outputBuffer
);
```

Parameters

resourceName
A string that specifies the resource on the server that the data will be posted to. Typically this is the name of an executable script on the server. The resource name should be specified using an absolute path that begins with a leading slash character.

patchData
A string that contains the patch data which will be provided to the server.

outputBuffer
A string passed by reference which will contain the reponse from the server when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PatchData** method is used to submit XML or JSON formatted patch data to a service, and then returns a copy of the response from the server into a string buffer. This method will not perform any encoding and will not automatically define the type of patch data being submitted. Your application is responsible for specifying the content type for the patch data and ensuring that the XML or JSON data being submitted to the server is formatted correctly.

This method sends a PATCH command to the server, which is similar to a POST or PUT request. It is used to make partial updates to a resource, rather than creating or replacing it entirely. The format of the patch data is specific to the service being used. If the resource being patched does not exist, the behavior is defined by the server. If enough information is provided, it may choose to create the resource just as if a PUT command was used, or it may return an error.

Your application should use the **SetHeader** method to define the Content-Type header prior to calling the **PatchData** method. One of the most common formats used is the JSON Merge Patch which is defined in RFC 7396. The value for the Content-Type header for this patch format is "application/merge-patch+json". Refer to your service API documentation to determine what patch formats are acceptable, along with any additional header values which must be defined.

This method will cause the current thread to block until the operation completes, a timeout occurs or the post is canceled. During the operation, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PatchData Overload List

HttpClient.PatchData Method (String, String)

Submits patch data to the server and returns the result in a string.

[Visual Basic]

```
Overloads Public Function PatchData( _
   ByVal patchData As String, _
   ByRef outputBuffer As String _
) As Boolean
```

[C#]

```
public bool PatchData(
   string patchData,
   ref string outputBuffer
);
```

Parameters

patchData
A string that contains the patch data which will be provided to the server.

outputBuffer
A string passed by reference which will contain the reponse from the server when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PatchData** method is used to submit XML or JSON formatted patch data to a service, and then returns a copy of the response from the server into a string buffer. This method will not perform any encoding and will not automatically define the type of patch data being submitted. Your application is responsible for specifying the content type for the patch data and ensuring that the XML or JSON data being submitted to the server is formatted correctly.

This method sends a PATCH command to the server, which is similar to a POST or PUT request. It is used to make partial updates to a resource, rather than creating or replacing it entirely. The format of the patch data is specific to the service being used. If the resource being patched does not exist, the behavior is defined by the server. If enough information is provided, it may choose to create the resource just as if a PUT command was used, or it may return an error.

Your application should use the **SetHeader** method to define the Content-Type header prior to calling the **PatchData** method. One of the most common formats used is the JSON Merge Patch which is defined in RFC 7396. The value for the Content-Type header for this patch format is "application/merge-patch+json". Refer to your service API documentation to determine what patch formats are acceptable, along with any additional header values which must be defined.

This version of the method submits the patch data to the resourse specified by the **Resource** property.

This method will cause the current thread to block until the operation completes, a timeout occurs or the post is canceled. During the operation, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PatchData Overload List

HttpClient.PostData Method

Submits the contents of the specified buffer to a resource on the server.

Overload List

Submits the contents of the specified buffer to a resource on the server.

public bool PostData(byte[],int,byte[],ref int);

Submits the contents of the specified buffer to a resource on the server.

public bool PostData(string,byte[],int,byte[],ref int);

Submits the contents of the specified buffer to a resource on the server.

public bool PostData(string,byte[],int,byte[],ref int,HttpPostOptions);

Submits the contents of the specified buffer to a resource on the server.

public bool PostData(string,string,ref string);

Submits the contents of the specified buffer to a resource on the server.

public bool PostData(string,string,ref string,HttpPostOptions);

Submits the contents of the specified buffer to a script on the server.

public bool PostData(string,ref string);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.PostData Method (String, String)

Submits the contents of the specified buffer to a script on the server.

[Visual Basic]

```
Overloads Public Function PostData( _
   ByVal inputBuffer As String, _
   ByRef outputBuffer As String _
) As Boolean
```

[C#]

```
public bool PostData(
   string inputBuffer,
   ref string outputBuffer
);
```

Parameters

inputBuffer

A string which contains the data that will be sent to the server. Unicode strings are automatically UTF-8 encoded prior to being submitted.

outputBuffer

A string passed by reference that will contain the output generated by the script. Typically this is HTML content which is generated by the script as a result of processing the data that was posted to it.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PostData** method submits the contents of the specified buffer to a script on the remote server and returns the result in a string provided by the caller. This method will cause the current thread to block until the operation completes, a timeout occurs or the post is canceled. During the operation, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PostData Overload List

HttpClient.PostData Method (String, String, String)

Submits the contents of the specified buffer to a resource on the server.

[Visual Basic]

```
Overloads Public Function PostData( _
   ByVal resourceName As String, _
   ByVal inputBuffer As String, _
   ByRef outputBuffer As String _
) As Boolean
```

[C#]

```
public bool PostData(
   string resourceName,
   string inputBuffer,
   ref string outputBuffer
);
```

Parameters

resourceName

A string that specifies the resource on the server that the data will be posted to. Typically this is the name of an executable script on the server. The resource name should be specified using an absolute path that begins with a leading slash character.

inputBuffer

A string which contains the data which will be sent to the server. Unicode strings are automatically UTF-8 encoded prior to being submitted.

outputBuffer

A string passed by reference that will contain the output generated by the script. Typically this is HTML content which is generated by the script as a result of processing the data that was posted to it.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PostData** method submits the contents of the specified buffer to a script on the remote server and returns the result in a string provided by the caller. This method will cause the current thread to block until the operation completes, a timeout occurs or the post is canceled. During the operation, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PostData Overload List

HttpClient.PostData Method (String, String, String, HttpPostOptions)

Submits the contents of the specified buffer to a resource on the server.

[Visual Basic]

```
Overloads Public Function PostData( _
   ByVal resourceName As String, _
   ByVal inputBuffer As String, _
   ByRef outputBuffer As String, _
   ByVal options As HttpPostOptions _
) As Boolean
```

[C#]

```
public bool PostData(
   string resourceName,
   string inputBuffer,
   ref string outputBuffer,
   HttpPostOptions options
);
```

Parameters

resourceName

A string that specifies the resource on the server that the data will be posted to. Typically this is the name of an executable script on the server. The resource name should be specified using an absolute path that begins with a leading slash character.

inputBuffer

A string which contains the data which will be sent to the server. Unicode strings are automatically UTF-8 encoded prior to being submitted.

outputBuffer

A string passed by reference that will contain the output generated by the script. Typically this is HTML content which is generated by the script as a result of processing the data that was posted to it.

options

An HttpPostOptions enumeration that specifies one or more options when posting data to the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PostData** method submits the contents of the specified buffer to a script on the remote server and returns the result in a string provided by the caller. This method will cause the current thread to block until the operation completes, a timeout occurs or the post is canceled. During the operation, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PostData Overload List

HttpClient.PostData Method (Byte[], Int32, Byte[], Int32)

Submits the contents of the specified buffer to a resource on the server.

[Visual Basic]

```
Overloads Public Function PostData( _
   ByVal inputBuffer As Byte(), _
   ByVal inputLength As Integer, _
   ByVal outputBuffer As Byte(), _
   ByRef outputLength As Integer _
) As Boolean
```

[C#]

```
public bool PostData(
   byte[] inputBuffer,
   int inputLength,
   byte[] outputBuffer,
   ref int outputLength
);
```

Parameters

inputBuffer
A byte array which contains the data to be submitted to the server.

inputLength
An integer value which specifies the size of the input buffer byte array. This value cannot be larger than the size of the buffer specified by the caller.

outputBuffer
A byte array that will contain the output generated by the script. Typically this is HTML content which is generated by the script as a result of processing the data that was posted to it.

outputLength
An integer value which specifies the maximum number of bytes of data to store in the output buffer. This value cannot be larger than the size of the buffer specified by the caller

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PostData** method submits the contents of the specified buffer to a script on the remote server and returns the result in a byte array provided by the caller. This method will cause the current thread to block until the operation completes, a timeout occurs or the post is canceled. During the operation, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PostData Overload List

HttpClient.PostData Method (String, Byte[], Int32, Byte[], Int32)

Submits the contents of the specified buffer to a resource on the server.

[Visual Basic]

```
Overloads Public Function PostData( _
   ByVal resourceName As String, _
   ByVal inputBuffer As Byte(), _
   ByVal inputLength As Integer, _
   ByVal outputBuffer As Byte(), _
   ByRef outputLength As Integer _
) As Boolean
```

[C#]

```
public bool PostData(
   string resourceName,
   byte[] inputBuffer,
   int inputLength,
   byte[] outputBuffer,
   ref int outputLength
);
```

Parameters

resourceName

A string that specifies the resource on the server that the data will be posted to. Typically this is the name of an executable script on the server. The resource name should be specified using an absolute path that begins with a leading slash character.

inputBuffer

A byte array which contains the data to be submitted to the server.

inputLength

An integer value which specifies the size of the input buffer byte array. This value cannot be larger than the size of the buffer specified by the caller.

outputBuffer

A byte array that will contain the output generated by the script. Typically this is HTML content which is generated by the script as a result of processing the data that was posted to it.

outputLength

An integer value which specifies the maximum number of bytes of data to store in the output buffer. This value cannot be larger than the size of the buffer specified by the caller

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PostData** method submits the contents of the specified buffer to a script on the remote server and returns the result in a byte array provided by the caller. This method will cause the current thread to block until the operation completes, a timeout occurs or the post is canceled. During the operation, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PostData Overload List

HttpClient.PostData Method (String, Byte[], Int32, Byte[], Int32, HttpPostOptions)

Submits the contents of the specified buffer to a resource on the server.

[Visual Basic]

```
Overloads Public Function PostData( _
   ByVal resourceName As String, _
   ByVal inputBuffer As Byte(), _
   ByVal inputLength As Integer, _
   ByVal outputBuffer As Byte(), _
   ByRef outputLength As Integer, _
   ByVal options As HttpPostOptions _
) As Boolean
```

[C#]

```
public bool PostData(
   string resourceName,
   byte[] inputBuffer,
   int inputLength,
   byte[] outputBuffer,
   ref int outputLength,
   HttpPostOptions options
);
```

Parameters

resourceName
A string that specifies the resource on the server that the data will be posted to. Typically this is the name of an executable script on the server. The resource name should be specified using an absolute path that begins with a leading slash character.

inputBuffer
A byte array which contains the data to be submitted to the server.

inputLength
An integer value which specifies the size of the input buffer byte array. This value cannot be larger than the size of the buffer specified by the caller.

outputBuffer
A byte array that will contain the output generated by the script. Typically this is HTML content which is generated by the script as a result of processing the data that was posted to it.

outputLength
An integer value which specifies the maximum number of bytes of data to store in the output buffer. This value cannot be larger than the size of the buffer specified by the caller

options
An HttpPostOptions enumeration that specifies one or more options when posting data to the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PostData** method submits the contents of the specified buffer to a script on the remote server and returns the result in a byte array provided by the caller. This method will cause the current thread to block until the operation completes, a timeout occurs or the post is canceled. During the operation, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PostData Overload List

HttpClient.PostFile Method

Post the contents of the specified file to a script executed on the remote server.

Overload List

Post the contents of the specified file to a script executed on the remote server.

public bool PostFile(string);

Upload the contents of a file to a resource on the server.

public bool PostFile(string,string);

Upload the contents of a file to a resource on the server.

public bool PostFile(string,string,string);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.PostFile Method (String)

Post the contents of the specified file to a script executed on the remote server.

[Visual Basic]

```
Overloads Public Function PostFile( _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool PostFile(
   string fileName
);
```

Parameters

fileName

A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PostFile** method posts the contents of a file to a script that is executed on the server. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The **PostFile** method is similar to the **PutFile** method in that it can be used to upload the contents of a local file to a remote server. However, instead of using the PUT command, the POST command is used to send the file data to a script that is executed on the server. This method has the advantage of not requiring any special configuration settings on the server, however it does require that the script be able to process multipart/form-data as defined in RFC 2388.

To support uploading files from a form on a webpage, the FILE input type is used along with the action that specifies the script that will accept the file data and process it. For example, the HTML code could look like this:

```
 <form action="/upload.php" method="post" enctype="multipart/form-data">
<input type="file" name="datafile" size="20"> <input type="submit">
</form>
```

In this example, the script /upload.php is responsible for processing the file data that is posted by the client, and the form field name "datafile" is used. The user can select a file, and when the Submit button is clicked, the file data is posted to the script. To simulate this using the **PostFile** method, the *LocalFile* argument should be set to the name of the local file that is to be posted to the server. The *Resource* argument should be the name of the script, in this case "/upload.php". The *FieldName* argument should be specified as the string "datafile" to match the name of the field used by the form.

Note that the **PostFile** function always submits the file contents as multipart/form-data with the content type set to application/octet-stream. The script that accepts the posted data must be able to parse the multipart header block and correctly process 8-bit data. If the script assumes that the data will be posted using a specific encoding type such as base64, then the file data may not be accepted or may be corrupted by the script.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PostFile Overload List

HttpClient.PostFile Method (String, String)

Upload the contents of a file to a resource on the server.

[Visual Basic]

```
Overloads Public Function PostFile( _
   ByVal fileName As String, _
   ByVal resourceName As String _
) As Boolean
```

[C#]

```
public bool PostFile(
   string fileName,
   string resourceName
);
```

Parameters

fileName
: A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

resourceName
: A string that specifies the resource on the server that the data will be posted to. Typically this is the name of an executable script on the server. The resource name should be specified using an absolute path that begins with a leading slash character.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PostFile** method posts the contents of a file to a script that is executed on the server. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The **PostFile** method is similar to the **PutFile** method in that it can be used to upload the contents of a local file to a remote server. However, instead of using the PUT command, the POST command is used to send the file data to a script that is executed on the server. This method has the advantage of not requiring any special configuration settings on the server, however it does require that the script be able to process multipart/form-data as defined in RFC 2388.

To support uploading files from a form on a webpage, the FILE input type is used along with the action that specifies the script that will accept the file data and process it. For example, the HTML code could look like this:

```
 <form action="/upload.php" method="post" enctype="multipart/form-data">
<input type="file" name="datafile" size="20"> <input type="submit">
</form>
```

In this example, the script /upload.php is responsible for processing the file data that is posted by the client, and the form field name "datafile" is used. The user can select a file, and when the Submit button is clicked, the file data is posted to the script. To simulate this using the **PostFile** method, the ***LocalFile*** argument should be set to the name of the local file that is to be posted to the server. The ***Resource*** argument should be the name of the script, in this case "/upload.php". The ***FieldName*** argument should be specified as the string "datafile" to match the name of the field used by the form.

Note that the **PostFile** function always submits the file contents as multipart/form-data with the content type set to application/octet-stream. The script that accepts the posted data must be able to parse the multipart header block and correctly process 8-bit data. If the script assumes that the data will be posted using a specific encoding type such as base64, then the file data may not be accepted or may be corrupted by the script.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PostFile Overload List

HttpClient.PostFile Method (String, String, String)

Upload the contents of a file to a resource on the server.

[Visual Basic]

```
Overloads Public Function PostFile( _
   ByVal fileName As String, _
   ByVal resourceName As String, _
   ByVal fieldName As String _
) As Boolean
```

[C#]

```
public bool PostFile(
   string fileName,
   string resourceName,
   string fieldName
);
```

Parameters

fileName

A string that specifies the file on the local system that will be transferred to the server. The file pathing and name conventions must be that of the local host.

resourceName

A string that specifies the resource on the server that the data will be posted to. Typically this is the name of an executable script on the server. The resource name should be specified using an absolute path that begins with a leading slash character.

fieldName

A string argument that specifies the form field name that the script expects. If this argument is omitted or is an empty string, a default field name of "File1" is used.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PostFile** method posts the contents of a file to a script that is executed on the server. This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The **PostFile** method is similar to the **PutFile** method in that it can be used to upload the contents of a local file to a remote server. However, instead of using the PUT command, the POST command is used to send the file data to a script that is executed on the server. This method has the advantage of not requiring any special configuration settings on the server, however it does require that the script be able to process multipart/form-data as defined in RFC 2388.

To support uploading files from a form on a webpage, the FILE input type is used along with the action that specifies the script that will accept the file data and process it. For example, the HTML code could look like this:

```
 <form action="/upload.php" method="post" enctype="multipart/form-data">
<input type="file" name="datafile" size="20"> <input type="submit">
</form>
```

In this example, the script /upload.php is responsible for processing the file data that is posted by the client, and the form field name "datafile" is used. The user can select a file, and when the Submit button is clicked, the file data is posted to the script. To simulate this using the **PostFile** method, the ***LocalFile*** argument should be set to the name of the local file that is to be posted to the server. The ***Resource*** argument should be the name of the script, in this case "/upload.php". The ***FieldName*** argument should be specified as the string "datafile" to match the name of the field used by the form.

Note that the **PostFile** function always submits the file contents as multipart/form-data with the content type set to application/octet-stream. The script that accepts the posted data must be able to parse the multipart header block and correctly process 8-bit data. If the script assumes that the data will be posted using a specific encoding type such as base64, then the file data may not be accepted or may be corrupted by the script.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PostFile Overload List

HttpClient.PutData Method

Transfer data from a local buffer to the server.

Overload List

Transfer data from a local buffer to the server.

public bool PutData(byte[],int);

Transfer data from a local buffer to the server.

public bool PutData(string);

Transfer data from a local buffer to the server.

public bool PutData(string,byte[],int);

Transfer data from a local buffer to the server.

public bool PutData(string,MemoryStream);

Transfer data from a local buffer to the server.

public bool PutData(string,string);

Transfer data from a local buffer to the server.

public bool PutData(string,string,int);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.PutData Method (String)

Transfer data from a local buffer to the server.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal buffer As String _
) As Boolean
```

[C#]

```
public bool PutData(
   string buffer
);
```

Parameters

buffer

A string which contains the data to be uploaded. Unicode strings are automatically UTF-8 encoded prior to being submitted.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutData** method transfers data from a local buffer and submits it to the server using the PUT command. This may be used to create a new file, replace the contents of an file which alread exists, or a request to submit data to a service API.

If you are using this method to upload the contents of a file, check to make sure the service allows the use of the PUT command. Not all servers permit files to be created using this method, and some may require specific configuration changes be made to the server to support this functionality.

The value of the **Resource** property will specify the resource path which will be used to submit the data.

It may be required for the client authenticate itself by setting the **UserName** and **Password** properties prior to submitting the data.

This method will cause the current thread to block until the data transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PutData Overload List

HttpClient.PutData Method (String, String)

Transfer data from a local buffer to the server.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal resourceName As String, _
   ByVal buffer As String _
) As Boolean
```

[C#]

```
public bool PutData(
   string resourceName,
   string buffer
);
```

Parameters

resourceName

A string that specifies the resource on the server that will receive the data being submitted. Depending on usage, this may be the name of a file or a URL which is used with an API.

buffer

A string which contains the data to be uploaded. Unicode strings are automatically UTF-8 encoded prior to being submitted.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutData** method transfers data from a local buffer and submits it to the server using the PUT command. This may be used to create a new file, replace the contents of an file which alread exists, or a request to submit data to a service API.

If you are using this method to upload the contents of a file, check to make sure the service allows the use of the PUT command. Not all servers permit files to be created using this method, and some may require specific configuration changes be made to the server to support this functionality.

It may be required for the client authenticate itself by setting the **UserName** and **Password** properties prior to submitting the data.

This method will cause the current thread to block until the data transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PutData Overload List

HttpClient.PutData Method (String, String, Int32)

Transfer data from a local buffer to the server.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal resourceName As String, _
   ByVal buffer As String, _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool PutData(
   string resourceName,
   string buffer,
   int length
);
```

Parameters

resourceName

A string that specifies the resource on the server that will receive the data being submitted. Depending on usage, this may be the name of a file or a URL which is used with an API.

buffer

A string which contains the data to be uploaded. Unicode strings are automatically UTF-8 encoded prior to being submitted.

length

An integer value which specifies the maximum number of bytes of data to sent. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutData** method transfers data from a local buffer and submits it to the server using the PUT command. This may be used to create a new file, replace the contents of an file which alread exists, or a request to submit data to a service API.

If you are using this method to upload the contents of a file, check to make sure the service allows the use of the PUT command. Not all servers permit files to be created using this method, and some may require specific configuration changes be made to the server to support this functionality.

It may be required for the client authenticate itself by setting the **UserName** and **Password** properties prior to submitting the data.

This method will cause the current thread to block until the data transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PutData Overload List

HttpClient.PutData Method (Byte[], Int32)

Transfer data from a local buffer to the server.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool PutData(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array which contains the data to be submitted.

length

An integer value which specifies the maximum number of bytes of data to sent. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutData** method transfers data from a local buffer and submits it to the server using the PUT command. This may be used to create a new file, replace the contents of an file which alread exists, or a request to submit data to a service API.

If you are using this method to upload the contents of a file, check to make sure the service allows the use of the PUT command. Not all servers permit files to be created using this method, and some may require specific configuration changes be made to the server to support this functionality.

The value of the **Resource** property will specify the resource path which will be used to submit the data.

It may be required for the client authenticate itself by setting the **UserName** and **Password** properties prior to submitting the data.

This method will cause the current thread to block until the data transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PutData Overload List

HttpClient.PutData Method (String, Byte[], Int32)

Transfer data from a local buffer to the server.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal resourceName As String, _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool PutData(
   string resourceName,
   byte[] buffer,
   int length
);
```

Parameters

resourceName
A string that specifies the resource on the server that will receive the data being submitted. Depending on usage, this may be the name of a file or a URL which is used with an API.

buffer
A byte array which contains the data to be submitted.

length
An integer value which specifies the maximum number of bytes of data to sent. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutData** method transfers data from a local buffer and submits it to the server using the PUT command. This may be used to create a new file, replace the contents of an file which alread exists, or a request to submit data to a service API.

If you are using this method to upload the contents of a file, check to make sure the service allows the use of the PUT command. Not all servers permit files to be created using this method, and some may require specific configuration changes be made to the server to support this functionality.

It may be required for the client authenticate itself by setting the **UserName** and **Password** properties prior to submitting the data.

This method will cause the current thread to block until the data transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PutData Overload List

HttpClient.PutData Method (String, MemoryStream)

Transfer data from a local buffer to the server.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal resourceName As String, _
   ByVal memStream As MemoryStream _
) As Boolean
```

[C#]

```
public bool PutData(
   string resourceName,
   MemoryStream memStream
);
```

Parameters

resourceName

A string that specifies the resource on the server that will receive the data being submitted. Depending on usage, this may be the name of a file or a URL which is used with an API.

memStream

A MemoryStream object which contains the data to be uploaded to the server. This stream must be open and readable.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutData** method transfers data from a local buffer and submits it to the server using the PUT command. This may be used to create a new file, replace the contents of an file which alread exists, or a request to submit data to a service API.

If you are using this method to upload the contents of a file, check to make sure the service allows the use of the PUT command. Not all servers permit files to be created using this method, and some may require specific configuration changes be made to the server to support this functionality.

It may be required for the client authenticate itself by setting the **UserName** and **Password** properties prior to submitting the data.

This method will cause the current thread to block until the data transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PutData Overload List

HttpClient.PostXml Method

Submits XML formatted data to the server and returns the result in a string.

Overload List

Submits XML formatted data to the server and returns the result in a string.

public bool PostXml(string,string,ref string);

Submits XML formatted data to the server and returns the result in a string.

public bool PostXml(string,string,ref string,HttpPostOptions);

Submits XML formatted data to the server and returns the result in a string.

public bool PostXml(string,ref string);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.PostXml Method (String, String)

Submits XML formatted data to the server and returns the result in a string.

[Visual Basic]

```
Overloads Public Function PostXml( _
   ByVal xmlData As String, _
   ByRef outputBuffer As String _
) As Boolean
```

[C#]

```
public bool PostXml(
   string xmlData,
   ref string outputBuffer
);
```

Parameters

xmlData

A string that contains the XML formatted data which will be provided to the script.

outputBuffer

A string passed by reference that will contain the output generated by the script. Typically this is HTML content which is generated by the script as a result of processing the data that was posted to it.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PostXml** method is used to submit XML formatted data to a script that executes on the server and then copy the output from that script into a local buffer. This function automatically sets the correct content type and encoding required for submitting XML data to a server, however it does not parse the XML data itself to ensure that it is well-formed. Your application is responsible for ensuring that the XML data that is being submitted to the server is formatted correctly.

This method will cause the current thread to block until the operation completes, a timeout occurs or the post is canceled. During the operation, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The XML data will be submitted to the server using the value of the **Resource** property to specify the script that will process the data.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PostXml Overload List | Resource Property | URL Property

HttpClient.PostXml Method (String, String, String)

Submits XML formatted data to the server and returns the result in a string.

[Visual Basic]

```
Overloads Public Function PostXml( _
   ByVal resourceName As String, _
   ByVal xmlData As String, _
   ByRef outputBuffer As String _
) As Boolean
```

[C#]

```
public bool PostXml(
   string resourceName,
   string xmlData,
   ref string outputBuffer
);
```

Parameters

resourceName
A string that specifies the resource on the server that the data will be posted to. Typically this is the name of an executable script on the server. The resource name should be specified using an absolute path that begins with a leading slash character.

xmlData
A string that contains the XML formatted data which will be provided to the script.

outputBuffer
A string passed by reference that will contain the output generated by the script. Typically this is HTML content which is generated by the script as a result of processing the data that was posted to it.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PostXml** method is used to submit XML formatted data to a script that executes on the server and then copy the output from that script into a local buffer. This function automatically sets the correct content type and encoding required for submitting XML data to a server, however it does not parse the XML data itself to ensure that it is well-formed. Your application is responsible for ensuring that the XML data that is being submitted to the server is formatted correctly.

This method will cause the current thread to block until the operation completes, a timeout occurs or the post is canceled. During the operation, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PostXml Overload List

HttpClient.PostXml Method (String, String, String, HttpPostOptions)

Submits XML formatted data to the server and returns the result in a string.

[Visual Basic]

```
Overloads Public Function PostXml( _
   ByVal resourceName As String, _
   ByVal xmlData As String, _
   ByRef outputBuffer As String, _
   ByVal options As HttpPostOptions _
) As Boolean
```

[C#]

```
public bool PostXml(
   string resourceName,
   string xmlData,
   ref string outputBuffer,
   HttpPostOptions options
);
```

Parameters

resourceName

A string that specifies the resource on the server that the data will be posted to. Typically this is the name of an executable script on the server. The resource name should be specified using an absolute path that begins with a leading slash character.

xmlData

A string that contains the XML formatted data which will be provided to the script.

outputBuffer

A string passed by reference that will contain the output generated by the script. Typically this is HTML content which is generated by the script as a result of processing the data that was posted to it.

options

An HttpPostOptions enumeration that specifies one or more options when posting data to the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PostXml** method is used to submit XML formatted data to a script that executes on the server and then copy the output from that script into a local buffer. This function automatically sets the correct content type and encoding required for submitting XML data to a server, however it does not parse the XML data itself to ensure that it is well-formed. Your application is responsible for ensuring that the XML data that is being submitted to the server is formatted correctly.

This method will cause the current thread to block until the operation completes, a timeout occurs or the post is canceled. During the operation, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PostXml Overload List

HttpClient.PutFile Method

Transfer a file from the local system to the web server.

Overload List

Transfer a file from the local system to the web server.

public bool PutFile(string);

Transfer a file from the local system to the web server.

public bool PutFile(string,HttpTransferOptions);

Transfer a file from the local system to the web server.

public bool PutFile(string,string);

Transfer a file from the local system to the web server.

public bool PutFile(string,string,HttpTransferOptions);

Transfer a file from the local system to the web server.

public bool PutFile(string,string,HttpTransferOptions,int);

Transfer a file from the local system to the web server.

public bool PutFile(string,string,HttpTransferOptions,long);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.PutFile Method (String, String)

Transfer a file from the local system to the web server.

[Visual Basic]

```
Overloads Public Function PutFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String _
) As Boolean
```

[C#]

```
public bool PutFile(
   string localFile,
   string remoteFile
);
```

Parameters

localFile

A string that specifies the file on the local system that will be transferred from the local system. The file pathing and name conventions must be that of the local host.

remoteFile

A string that specifies the file on the server that will contain the data being transferred. If the file already exists, it will be overwritten. The file name should be specified using an absolute path that begins with a leading slash character.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutFile** method is used to transfer a file from the local system to a remote server. Not all servers permit files to be uploaded using this method, and some may require that specific configuration changes be made to the server in order to support this functionality. Consult your server's technical reference documentation to see if it supports the PUT command, and if so, what must be done to enable it. It may be required that the client authenticate itself by setting the **UserName** and **Password** properties prior to uploading the file.

This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PutFile Overload List

HttpClient.PutFile Method (String, HttpTransferOptions)

Transfer a file from the local system to the web server.

[Visual Basic]

```
Overloads Public Function PutFile( _
   ByVal localFile As String, _
   ByVal options As HttpTransferOptions _
) As Boolean
```

[C#]

```
public bool PutFile(
   string localFile,
   HttpTransferOptions options
);
```

Parameters

localFile
: A string that specifies the file on the local system that will be transferred from the local system. The file pathing and name conventions must be that of the local host.

options
: An HttpTransferOptions enumeration value which specifies one or more file transfer options.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutFile** method is used to transfer a file from the local system to a remote server. Not all servers permit files to be uploaded using this method, and some may require that specific configuration changes be made to the server in order to support this functionality. Consult your server's technical reference documentation to see if it supports the PUT command, and if so, what must be done to enable it. It may be required that the client authenticate itself by setting the **UserName** and **Password** properties prior to uploading the file.

This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PutFile Overload List

HttpClient.PutFile Method (String, String, HttpTransferOptions)

Transfer a file from the local system to the web server.

[Visual Basic]

```
Overloads Public Function PutFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal options As HttpTransferOptions _
) As Boolean
```

[C#]

```
public bool PutFile(
   string localFile,
   string remoteFile,
   HttpTransferOptions options
);
```

Parameters

localFile
A string that specifies the file on the local system that will be transferred from the local system. The file pathing and name conventions must be that of the local host.

remoteFile
A string that specifies the file on the server that will contain the data being transferred. If the file already exists, it will be overwritten. The file name should be specified using an absolute path that begins with a leading slash character.

options
An HttpTransferOptions enumeration value which specifies one or more file transfer options.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutFile** method is used to transfer a file from the local system to a remote server. Not all servers permit files to be uploaded using this method, and some may require that specific configuration changes be made to the server in order to support this functionality. Consult your server's technical reference documentation to see if it supports the PUT command, and if so, what must be done to enable it. It may be required that the client authenticate itself by setting the **UserName** and **Password** properties prior to uploading the file.

This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PutFile Overload List

HttpClient.PutFile Method (String, String, HttpTransferOptions, Int64)

Transfer a file from the local system to the web server.

[Visual Basic]

```
Overloads Public Function PutFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal options As HttpTransferOptions, _
   ByVal offset As Long _
) As Boolean
```

[C#]

```
public bool PutFile(
   string localFile,
   string remoteFile,
   HttpTransferOptions options,
   long offset
);
```

Parameters

localFile
A string that specifies the file on the local system that will be transferred from the local system. The file pathing and name conventions must be that of the local host.

remoteFile
A string that specifies the file on the server that will contain the data being transferred. If the file already exists, it will be overwritten. The file name should be specified using an absolute path that begins with a leading slash character.

options
An HttpTransferOptions enumeration value which specifies one or more file transfer options.

offset
A integer value which specifies the offset where the file transfer should begin. A value of zero specifies that the file transfer should start at the beginning of the file. A value greater than zero is typically used to restart a transfer that has not completed successfully. Note that specifying a non-zero offset requires that the server support the option to restart file transfers.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutFile** method is used to transfer a file from the local system to a remote server. Not all servers permit files to be uploaded using this method, and some may require that specific configuration changes be made to the server in order to support this functionality. Consult your server's technical reference documentation to see if it supports the PUT command, and if so, what must be done to enable it. It may be required that the client authenticate itself by setting the **UserName** and **Password** properties prior to uploading the file.

This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PutFile Overload List

HttpClient.PutFile Method (String, String, HttpTransferOptions, Int32)

Transfer a file from the local system to the web server.

[Visual Basic]

```
Overloads Public Function PutFile( _
   ByVal localFile As String, _
   ByVal remoteFile As String, _
   ByVal options As HttpTransferOptions, _
   ByVal offset As Integer _
) As Boolean
```

[C#]

```
public bool PutFile(
   string localFile,
   string remoteFile,
   HttpTransferOptions options,
   int offset
);
```

Parameters

localFile

A string that specifies the file on the local system that will be transferred from the local system. The file pathing and name conventions must be that of the local host.

remoteFile

A string that specifies the file on the server that will contain the data being transferred. If the file already exists, it will be overwritten. The file name should be specified using an absolute path that begins with a leading slash character.

options

An HttpTransferOptions enumeration value which specifies one or more file transfer options.

offset

A integer value which specifies the offset where the file transfer should begin. A value of zero specifies that the file transfer should start at the beginning of the file. A value greater than zero is typically used to restart a transfer that has not completed successfully. Note that specifying a non-zero offset requires that the server support the option to restart file transfers.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PutFile** method is used to transfer a file from the local system to a remote server. Not all servers permit files to be uploaded using this method, and some may require that specific configuration changes be made to the server in order to support this functionality. Consult your server's technical reference documentation to see if it supports the PUT command, and if so, what must be done to enable it. It may be required that the client authenticate itself by setting the **UserName** and **Password** properties prior to uploading the file.

This method will cause the current thread to block until the file transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.PutFile Overload List

HttpClient.Read Method

Read data from the server and store it in a byte array.

Overload List

Read data from the server and store it in a byte array.

public int Read(byte[]);

Read data from the server and store it in a byte array.

public int Read(byte[],int);

Read data from the server and store it in a string.

public int Read(ref string);

Read data from the server and store it in a string.

public int Read(ref string,int);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.Read Method (Byte[])

Read data from the server and store it in a byte array.

[Visual Basic]

```
Overloads Public Function Read( _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Read(
   byte[] buffer
);
```

Parameters

buffer

A byte array that the data will be stored in.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the size of the byte array passed to the method. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Read Overload List

HttpClient.Read Method (Byte[], Int32)

Read data from the server and store it in a byte array.

[Visual Basic]

```
Overloads Public Function Read( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Read(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array that the data will be stored in.

length

An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the number of bytes specified. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Read Overload List

HttpClient.Read Method (String)

Read data from the server and store it in a string.

[Visual Basic]

```
Overloads Public Function Read( _
   ByRef buffer As String _
) As Integer
```

[C#]

```
public int Read(
   ref string buffer
);
```

Parameters

buffer

A string that will contain the data read from the client.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to a maximum of 4096 bytes. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Read Overload List

HttpClient.Read Method (String, Int32)

Read data from the server and store it in a string.

[Visual Basic]

```
Overloads Public Function Read( _
   ByRef buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Read(
   ref string buffer,
   int length
);
```

Parameters

buffer
A string that will contain the data read from the client.

length
An integer value which specifies the maximum number of bytes of data to read.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the number of bytes specified. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Read Overload List

HttpClient.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. If a socket has been allocated, it will be released and any active connections will be terminated. All properties will be reset to their default values.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.SetCookie Method

Send the specified cookie to the server when a resource is requested.

[Visual Basic]

```
Public Function SetCookie( _
   ByVal cookieName As String, _
   ByVal cookieValue As String _
) As Boolean
```

[C#]

```
public bool SetCookie(
   string cookieName,
   string cookieValue
);
```

Parameters

cookieName

A string which specifies the name of the cookie that will be sent to the server when the next resource is requested.

cookieValue

A string which specifies the value of the cookie. To delete a cookie that has been previously set, this parameter should be an empty string.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SetCookie** method submits the cookie name and value to the server when a resource is requested or data is posted to a script. For more information about cookies and how they are used, refer to the **GetCookie** method.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.SetHeader Method

Set the value of a request header field.

[Visual Basic]

```
Public Function SetHeader( _
   ByVal headerName As String, _
   ByVal headerValue As String _
) As Boolean
```

[C#]

```
public bool SetHeader(
   string headerName,
   string headerValue
);
```

Parameters

headerName
A string that specifies the name of the request header.

headerValue
A string that specifies the value associated with the request header. If this argument is set to an empty string, the request header is and its previous value are deleted.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SetHeader** method is used to set the values of specific fields in the HTTP request header. This method should be called before the client has requested the resource.

Some headers are generated by methods that send resource requests. Some of these are supplied by the requesting methods only if the application has not previously defined the header. For others, the requesting method overrides what the application may have defined. The affected headers include:

Header	Description
Accept	This header field specifies the type of content that the client will accept. By default, all content types will be accepted by the client.
Authorization	This header field specifies the authentication information required to access the resource. By default, this header field is only defined if the client specifies a username and password.
Connection	This header field is automatically defined based on the value specified by the **KeepAlive** property.
Content-Length	This header field is automatically defined when data is being posted to the server or a file is being uploaded.
Content-Type	This header field is automatically defined when data is being posted to the server or a file is being

	uploaded.
Host	This header field defines the hostname of the server. By default, the hostname provided to the **Connect** method will be used.
Proxy-Authorization	This header field specifies the proxy authentication information required to access the resource. By default, this header field is only defined if a proxy server has been specified, along with a username and password.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.SubmitForm Method

Submits the current form data to the server for processing.

Overload List

Submits the current form data to the server for processing.

public bool SubmitForm();

Submits the current form data to the server for processing.

public bool SubmitForm(byte[],ref int);

Submits the current form data to the server for processing.

public bool SubmitForm(byte[],ref int,HttpSubmitOptions);

Submits the current form data to the server for processing.

public bool SubmitForm(ref string);

Submits the current form data to the server for processing.

public bool SubmitForm(ref string,ref int);

Submits the current form data to the server for processing.

public bool SubmitForm(ref string,ref int,HttpSubmitOptions);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.SubmitForm Method ()

Submits the current form data to the server for processing.

[Visual Basic]

```
Overloads Public Function SubmitForm() As Boolean
```

[C#]

```
public bool SubmitForm();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SubmitForm** method submits the current form data to a script on the remote server and returns the result in a byte array provided by the caller. This method will cause the current thread to block until the operation completes, a timeout occurs or the post is canceled. During the operation, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Any data returned by the server will be discarded.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.SubmitForm Overload List

HttpClient.SubmitForm Method (Byte[], Int32)

Submits the current form data to the server for processing.

[Visual Basic]

```
Overloads Public Function SubmitForm( _
   ByVal resultBuffer As Byte(), _
   ByRef resultLength As Integer _
) As Boolean
```

[C#]

```
public bool SubmitForm(
   byte[] resultBuffer,
   ref int resultLength
);
```

Parameters

resultBuffer
A byte array which will contain the data returned by the server.

resultLength
An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SubmitForm** method submits the current form data to a script on the remote server and returns the result in a byte array provided by the caller. This method will cause the current thread to block until the operation completes, a timeout occurs or the post is canceled. During the operation, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.SubmitForm Overload List

HttpClient.SubmitForm Method (Byte[], Int32, HttpSubmitOptions)

Submits the current form data to the server for processing.

[Visual Basic]

```
Overloads Public Function SubmitForm( _
   ByVal resultBuffer As Byte(), _
   ByRef resultLength As Integer, _
   ByVal options As HttpSubmitOptions _
) As Boolean
```

[C#]

```
public bool SubmitForm(
   byte[] resultBuffer,
   ref int resultLength,
   HttpSubmitOptions options
);
```

Parameters

resultBuffer
A byte array which will contain the data returned by the server.

resultLength
An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

options
An HttpSubmitOptions enumeration value which specifies one or more options for submitting form data to the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SubmitForm** method submits the current form data to a script on the remote server and returns the result in a byte array provided by the caller. This method will cause the current thread to block until the operation completes, a timeout occurs or the post is canceled. During the operation, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.SubmitForm Overload List

HttpClient.SubmitForm Method (String)

Submits the current form data to the server for processing.

[Visual Basic]

```
Overloads Public Function SubmitForm( _
   ByRef resultBuffer As String _
) As Boolean
```

[C#]

```
public bool SubmitForm(
   ref string resultBuffer
);
```

Parameters

resultBuffer
A string passed by reference which will contain the data returned by the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SubmitForm** method submits the current form data to a script on the remote server and returns the result in a string provided by the caller. This method will cause the current thread to block until the operation completes, a timeout occurs or the post is canceled. During the operation, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.SubmitForm Overload List

HttpClient.SubmitForm Method (String, Int32)

Submits the current form data to the server for processing.

[Visual Basic]

```
Overloads Public Function SubmitForm( _
   ByRef resultBuffer As String, _
   ByRef resultLength As Integer _
) As Boolean
```

[C#]

```
public bool SubmitForm(
   ref string resultBuffer,
   ref int resultLength
);
```

Parameters

resultBuffer

A string passed by reference which will contain the data returned by the server.

resultLength

An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SubmitForm** method submits the current form data to a script on the remote server and returns the result in a string provided by the caller. This method will cause the current thread to block until the operation completes, a timeout occurs or the post is canceled. During the operation, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.SubmitForm Overload List

HttpClient.SubmitForm Method (String, Int32, HttpSubmitOptions)

Submits the current form data to the server for processing.

[Visual Basic]

```
Overloads Public Function SubmitForm( _
   ByRef resultBuffer As String, _
   ByRef resultLength As Integer, _
   ByVal options As HttpSubmitOptions _
) As Boolean
```

[C#]

```
public bool SubmitForm(
   ref string resultBuffer,
   ref int resultLength,
   HttpSubmitOptions options
);
```

Parameters

resultBuffer
A string passed by reference which will contain the data returned by the server.

resultLength
An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

options
An HttpSubmitOptions enumeration value which specifies one or more options for submitting form data to the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SubmitForm** method submits the current form data to a script on the remote server and returns the result in a string provided by the caller. This method will cause the current thread to block until the operation completes, a timeout occurs or the post is canceled. During the operation, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.SubmitForm Overload List

HttpClient.TaskAbort Method

Abort all asynchronous tasks that are currently active.

Overload List

Abort all asynchronous tasks that are currently active.

public bool TaskAbort();

Abort the specified asynchronous task.

public bool TaskAbort(int);

Abort the specified asynchronous task.

public bool TaskAbort(int,int);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.TaskAbort Method ()

Abort all asynchronous tasks that are currently active.

[Visual Basic]

```
Overloads Public Function TaskAbort() As Boolean
```

[C#]

```
public bool TaskAbort();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskAbort** method signals all background worker threads created by this instance of the class to abort their current operation and terminate as soon as possible. This version of the method will signal each active task and return immediately to the caller.

The **Reset** and **Uninitialize** methods will abort all active background transfers and wait for those tasks to complete before returning to the caller. It is recommended that your application explicitly wait for background transfers to complete or abort them using this method before allowing the program to terminate. This will ensure that your program can perform any necessary cleanup operations. If there are active background tasks running at the time that the class instance is disposed, it can force the instance to stop those worker threads immediately without waiting for them to terminate gracefully.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.TaskAbort Overload List

HttpClient.TaskAbort Method (Int32)

Abort the specified asynchronous task.

[Visual Basic]

```
Overloads Public Function TaskAbort( _
   ByVal taskId As Integer _
) As Boolean
```

[C#]

```
public bool TaskAbort(
   int taskId
);
```

Parameters

taskId

An integer value that specifies the unique identifier associated with a background task.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskAbort** method signals the background worker thread associated with the task ID to abort the current operation and terminate as soon as possible. This version of the method returns immediately after the background thread has been signaled.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.TaskAbort Overload List

HttpClient.TaskAbort Method (Int32, Int32)

Abort the specified asynchronous task.

[Visual Basic]

```
Overloads Public Function TaskAbort( _
   ByVal taskId As Integer, _
   ByVal timeWait As Integer _
) As Boolean
```

[C#]

```
public bool TaskAbort(
   int taskId,
   int timeWait
);
```

Parameters

taskId

An integer value that specifies the unique identifier associated with a background task.

timeWait

An integer value that specifies the number of milliseconds to wait for the background task to abort.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskAbort** method signals the background worker thread associated with the task ID to abort the current operation and terminate as soon as possible. If the ***timeWait*** parameter has a value of zero, the method returns immediately after the background thread has been signaled. If the ***timeWait*** parameter is non-zero, the method will wait that amount of time for the background thread to terminate.

The **Reset** and **Uninitialize** methods will abort all active background transfers and wait for those tasks to complete before returning to the caller. It is recommended that your application explicitly wait for background transfers to complete or abort them using this method before allowing the program to terminate. This will ensure that your program can perform any necessary cleanup operations. If there are active background tasks running at the time that the class instance is disposed, it can force the instance to stop those worker threads immediately without waiting for them to terminate gracefully.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.TaskAbort Overload List

HttpClient.TaskDone Method

Determine if the current asynchronous task has completed.

Overload List

Determine if the current asynchronous task has completed.

public bool TaskDone();

Determine if an asynchronous task has completed.

public bool TaskDone(int);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.TaskDone Method ()

Determine if the current asynchronous task has completed.

[Visual Basic]

```
Overloads Public Function TaskDone() As Boolean
```

[C#]

```
public bool TaskDone();
```

Return Value

This method returns a Boolean value. If the task has finished, the return value is **true**. If the background task is still active, the return value is **false**.

Remarks

The **TaskDone** method is used to determine if the current asynchronous task has completed. This overloaded version of the method is functionally equivalent to providing the value of the **TaskId** property as the unique task identifier.

If you use this method to poll the status of a background task from within the main UI thread, you must ensure that Windows messages are processed so that the application remains responsive to the end-user. To check if a background transfer has completed, it is recommended that you use a timer to periodically call this method rather than calling it repeatedly within a loop.

To determine if the task completed successfully, the **TaskWait** method will provide the last error code associated with the task. Note that if this method returns **true**, it is guaranteed that calling **TaskWait** using the same task ID will return the error code to the caller immediately without causing the current thread to block.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.TaskDone Overload List

HttpClient.TaskDone Method (Int32)

Determine if an asynchronous task has completed.

[Visual Basic]

```
Overloads Public Function TaskDone( _
   ByVal taskId As Integer _
) As Boolean
```

[C#]

```
public bool TaskDone(
   int taskId
);
```

Parameters

taskId

An optional integer value that specifies the unique identifier associated with a background task.

Return Value

This method returns a Boolean value. If the task has finished, the return value is **true**. If the background task is still active, the return value is **false**.

Remarks

The **TaskDone** method is used to determine if the specified asynchronous task has completed.

If you use this method to poll the status of a background task from within the main UI thread, you must ensure that Windows messages are processed so that the application remains responsive to the end-user. To check if a background transfer has completed, it is recommended that you use a timer to periodically call this method rather than calling it repeatedly within a loop.

To determine if the task completed successfully, the **TaskWait** method will provide the last error code associated with the task. Note that if this method returns **true**, it is guaranteed that calling **TaskWait** using the same task ID will return the error code to the caller immediately without causing the current thread to block.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.TaskDone Overload List

HttpClient.TaskResume Method

Resume execution of the current asynchronous task.

Overload List

Resume execution of the current asynchronous task.

public bool TaskResume();

Resume execution of an asynchronous task.

public bool TaskResume(int);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.TaskResume Method ()

Resume execution of the current asynchronous task.

[Visual Basic]
```
Overloads Public Function TaskResume() As Boolean
```

[C#]
```
public bool TaskResume();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskResume** method resumes execution of the current background task that was previously suspended using the **TaskSuspend** method. This overloaded version of the method is functionally equivalent to providing the value of the **TaskId** property as the unique task identifier.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.TaskResume Overload List

HttpClient.TaskResume Method (Int32)

Resume execution of an asynchronous task.

[Visual Basic]

```
Overloads Public Function TaskResume( _
   ByVal taskId As Integer _
) As Boolean
```

[C#]

```
public bool TaskResume(
   int taskId
);
```

Parameters

taskId

An integer value that specifies the unique identifier associated with a background task.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskResume** method resumes execution of the background worker thread that was previously suspended using the **TaskSuspend** method.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.TaskResume Overload List

HttpClient.TaskSuspend Method

Suspend execution of the current asynchronous task.

Overload List

Suspend execution of the current asynchronous task.

public bool TaskSuspend();

Suspend execution of an asynchronous task.

public bool TaskSuspend(int);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.TaskSuspend Method ()

Suspend execution of the current asynchronous task.

[Visual Basic]
```
Overloads Public Function TaskSuspend() As Boolean
```

[C#]
```
public bool TaskSuspend();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskSuspend** method will suspend execution of the background worker thread associated with the current task. This overloaded version of the method is functionally equivalent to providing the value of the **TaskId** property as the unique task identifier.

Once the task has been suspended, it will no longer be scheduled for execution, however the client session will remain active and the task may be resumed using the **TaskResume** method. Note that if a task is suspended for a long period of time, the background operation may fail because it has exceeded the timeout period imposed by the server.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.TaskSuspend Overload List

HttpClient.TaskSuspend Method (Int32)

Suspend execution of an asynchronous task.

[Visual Basic]

```
Overloads Public Function TaskSuspend( _
   ByVal taskId As Integer _
) As Boolean
```

[C#]

```
public bool TaskSuspend(
   int taskId
);
```

Parameters

taskId

An integer value that specifies the unique identifier associated with a background task.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskSuspend** method will suspend execution of the background worker thread associated with the task.

Once the task has been suspended, it will no longer be scheduled for execution, however the client session will remain active and the task may be resumed using the **TaskResume** method. Note that if a task is suspended for a long period of time, the background operation may fail because it has exceeded the timeout period imposed by the server.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.TaskSuspend Overload List

HttpClient.TaskWait Method

Wait for all asynchronous tasks to complete.

Overload List

Wait for all asynchronous tasks to complete.

public bool TaskWait();

Wait for an asynchronous task to complete.

public bool TaskWait(int);

Wait for an asynchronous task to complete.

public bool TaskWait(int,int);

Wait for an asynchronous task to complete.

public bool TaskWait(int,int,ref ErrorCode);

Wait for an asynchronous task to complete.

public bool TaskWait(int,int,ref int,ref ErrorCode);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.TaskWait Method ()

Wait for all asynchronous tasks to complete.

[Visual Basic]

```
Overloads Public Function TaskWait() As Boolean
```

[C#]

```
public bool TaskWait();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This overloaded version of the **TaskWait** method will cause the current working thread to block until all background tasks created by this instance of the class have completed. If there are no active background tasks, this method will return to the caller immediately.

You should not call this version of the method from the main UI thread. Windows messages will not be processed while this method is blocked waiting for the background tasks to complete, and this can cause your application to appear non-responsive to the end-user. If you have a GUI application and you need to determine if all tasks have completed, create a timer to periodically check the value of the **TaskCount** property. When it returns zero, there are no active background tasks executing.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.TaskWait Overload List

HttpClient.TaskWait Method (Int32)

Wait for an asynchronous task to complete.

[Visual Basic]

```
Overloads Public Function TaskWait( _
   ByVal taskId As Integer _
) As Boolean
```

[C#]

```
public bool TaskWait(
   int taskId
);
```

Parameters

taskId

An integer value that specifies the unique identifier associated with a background task.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskWait** method waits for the specified task to complete. This method will cause the current working thread to block for an indefinite period of time until the task completes. If the specified task has already completed at the time this method is called, the method will return immediately without causing the current thread to block.

You should not call this overloaded version of the method from the main UI thread. Windows messages will not be processed while this method is blocked waiting for the background task to complete, and this can cause your application to appear non-responsive to the end-user. If you have a GUI application and you need to determine if a background task has finished, create a timer to periodically call the **TaskDone** method. When it returns **true** (indicating that the task has completed), you can safely call **TaskWait** to obtain the elapsed time and last error code without blocking the current thread.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.TaskWait Overload List

HttpClient.TaskWait Method (Int32, Int32)

Wait for an asynchronous task to complete.

[Visual Basic]

```
Overloads Public Function TaskWait( _
   ByVal taskId As Integer, _
   ByVal timeWait As Integer _
) As Boolean
```

[C#]

```
public bool TaskWait(
   int taskId,
   int timeWait
);
```

Parameters

taskId

An integer value that specifies the unique identifier associated with a background task.

timeWait

An integer value that specifies the number of milliseconds to wait for the background task to complete.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskWait** method waits for the specified task to complete. This method will cause the current working thread to block until the task completes or the amount of time exceeds the number of milliseconds specified by the caller. If the *timeWait* parameter is zero, then this method will poll the status of the task and return immediately to the caller. If the specified task has already completed at the time this method is called, the method will return immediately without causing the current thread to block.

You should not call this method from the main UI thread with a long timeout period to wait for a background task to complete. Windows messages will not be processed while this method is blocked waiting for the background task to complete, and this can cause your application to appear non-responsive to the end-user. If you have a GUI application and you need to determine if a background task has finished, create a timer to periodically call the **TaskDone** method. When it returns **true** (indicating that the task has completed), you can safely call **TaskWait** to obtain the elapsed time and last error code without blocking the current thread.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.TaskWait Overload List

HttpClient.TaskWait Method (Int32, Int32, ErrorCode)

Wait for an asynchronous task to complete.

[Visual Basic]

```
Overloads Public Function TaskWait( _
   ByVal taskId As Integer, _
   ByVal timeWait As Integer, _
   ByRef taskError As ErrorCode _
) As Boolean
```

[C#]

```
public bool TaskWait(
   int taskId,
   int timeWait,
   ref ErrorCode taskError
);
```

Parameters

taskId
An integer value that specifies the unique identifier associated with a background task.

timeWait
An integer value that specifies the number of milliseconds to wait for the background task to complete.

taskError
An ErrorCode value passed by reference that will contain the last error code set by the asynchronous task.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskWait** method waits for the specified task to complete. This method will cause the current working thread to block until the task completes or the amount of time exceeds the number of milliseconds specified by the caller. If the *timeWait* parameter is zero, then this method will poll the status of the task and return immediately to the caller.

If the specified task has already completed at the time this method is called, the method will return immediately without causing the current thread to block. The *taskError* parameter will contain the last error code value that was set by the worker thread before it terminated. If the *taskError* value is zero, that means that the background task was successful and no error occurred. A non-zero value will indicate that the background task has failed.

You should not call this method from the main UI thread with a long timeout period to wait for a background task to complete. Windows messages will not be processed while this method is blocked waiting for the background task to complete, and this can cause your application to appear non-responsive to the end-user. If you have a GUI application and you need to determine if a background task has finished, create a timer to periodically call the **TaskDone** method. When it returns **true** (indicating that the task has completed), you can safely call **TaskWait** to obtain the elapsed time and last error code without blocking the current thread.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.TaskWait Overload List

HttpClient.TaskWait Method (Int32, Int32, Int32, ErrorCode)

Wait for an asynchronous task to complete.

[Visual Basic]

```
Overloads Public Function TaskWait( _
   ByVal taskId As Integer, _
   ByVal timeWait As Integer, _
   ByRef timeElapsed As Integer, _
   ByRef taskError As ErrorCode _
) As Boolean
```

[C#]

```
public bool TaskWait(
   int taskId,
   int timeWait,
   ref int timeElapsed,
   ref ErrorCode taskError
);
```

Parameters

taskId
An integer value that specifies the unique identifier associated with a background task.

timeWait
An integer value that specifies the number of milliseconds to wait for the background task to complete.

timeElapsed
An integer value passed by reference that will contain the elapsed time for the task in milliseconds when the method returns.

taskError
An ErrorCode value passed by reference that will contain the last error code set by the asynchronous task.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **TaskWait** method waits for the specified task to complete. This method will cause the current working thread to block until the task completes or the amount of time exceeds the number of milliseconds specified by the caller. If the *timeWait* parameter is zero, then this method will poll the status of the task and return immediately to the caller.

If the specified task has already completed at the time this method is called, the method will return immediately without causing the current thread to block. The *timeElapsed* parameter contain the number of milliseconds that it took for the task to complete. The *taskError* parameter will contain the last error code value that was set by the worker thread before it terminated. If the *taskError* value is zero, that means that the background task was successful and no error occurred. A non-zero value will indicate that the background task has failed.

You should not call this method from the main UI thread with a long timeout period to wait for a background task to complete. Windows messages will not be processed while this method is blocked

waiting for the background task to complete, and this can cause your application to appear non-responsive to the end-user. If you have a GUI application and you need to determine if a background task has finished, create a timer to periodically call the **TaskDone** method. When it returns **true** (indicating that the task has completed), you can safely call **TaskWait** to obtain the elapsed time and last error code without blocking the current thread.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.TaskWait Overload List

HttpClient.Uninitialize Method

Uninitialize the class library and release any resources allocated for the current thread.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method terminates any active connection, releases resources allocated for the current thread and unloads the networking library. After this method has been called, no further client operations may be performed until the class instance has been re-initialized.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the class is no longer needed.

See Also

HttpClient Class | SocketTools Namespace | Initialize Method

HttpClient.Write Method

Write one or more bytes of data to the server.

Overload List

Write one or more bytes of data to the server.

public int Write(byte[]);

Write one or more bytes of data to the server.

public int Write(byte[],int);

Write a string of characters to the server.

public int Write(string);

Write a string of characters to the server.

public int Write(string,int);

See Also

HttpClient Class | SocketTools Namespace

HttpClient.Write Method (Byte[])

Write one or more bytes of data to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Write(
   byte[] buffer
);
```

Parameters

buffer

A byte array that contains the data to be written to the server.

Return Value

An integer value which specifies the number of bytes actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Write Overload List

HttpClient.Write Method (Byte[], Int32)

Write one or more bytes of data to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Write(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array that contains the data to be written to the server.

length

An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Write Overload List

HttpClient.Write Method (String)

Write a string of characters to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String _
) As Integer
```

[C#]

```
public int Write(
   string buffer
);
```

Parameters

buffer

A string which contains the data to be written to the server.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Write Overload List

HttpClient.Write Method (String, Int32)

Write a string of characters to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Write(
   string buffer,
   int length
);
```

Parameters

buffer

A string which contains the data to be written to the server.

length

An integer value which specifies the maximum number of characters to write. This value cannot be larger than the length of the string specified by the caller.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

HttpClient Class | SocketTools Namespace | HttpClient.Write Overload List

HttpClient Events

The events of the **HttpClient** class are listed below. For a complete list of **HttpClient** class members, see the HttpClient Members topic.

Public Instance Events

⚡ OnCancel	Occurs when a blocking client operation is canceled.
⚡ OnCommand	Occurs when the client sends a command to the remote host and receives a reply indicating the result of that command.
⚡ OnConnect	Occurs when a connection is established with the remote host.
⚡ OnDisconnect	Occurs when the remote host disconnects from the local system.
⚡ OnError	Occurs when an client operation fails.
⚡ OnProgress	Occurs as a data stream is being read or written to the client.
⚡ OnRead	Occurs when data is available to be read from the client.
⚡ OnRedirect	Occurs when the server indicates a resource has been moved.
⚡ OnTaskBegin	Occurs when an asynchronous task begins execution.
⚡ OnTaskEnd	Occurs when an asynchronous task completes.
⚡ OnTaskRun	Occurs while a background task is active.
⚡ OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
⚡ OnWrite	Occurs when data can be written to the client.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.OnCancel Event

Occurs when a blocking client operation is canceled.

[Visual Basic]

```
Public Event OnCancel As EventHandler
```

[C#]

```
public event EventHandler OnCancel;
```

Remarks

The **OnCancel** event is generated when a blocking client operation, such as sending or receiving data, is canceled with the **Cancel** method. To assist in determining which operation was canceled, check the value of the **Status** property.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.OnCommand Event

Occurs when the client sends a command to the remote host and receives a reply indicating the result of that command.

[Visual Basic]

```
Public Event OnCommand As OnCommandEventHandler
```

[C#]

```
public event OnCommandEventHandler OnCommand;
```

Event Data

The event handler receives an argument of type HttpClient.CommandEventArgs containing data related to this event. The following **HttpClient.CommandEventArgs** properties provide information specific to this event.

Property	Description
ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.

Remarks

The **OnCommand** event is generated when the client receives a reply from the server after some action has been taken.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.CommandEventArgs Class

Provides data for the OnCommand event.

For a list of all members of this type, see HttpClient.CommandEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.HttpClient.CommandEventArgs

[Visual Basic]
```
Public Class HttpClient.CommandEventArgs
    Inherits EventArgs
```

[C#]
```
public class HttpClient.CommandEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

CommandEventArgs specifies the result code and result string for the last command executed by the server.

The OnCommand event occurs whenever a command is executed on the server.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

HttpClient.CommandEventArgs Members | SocketTools Namespace

HttpClient.CommandEventArgs Members

HttpClient.CommandEventArgs overview

Public Instance Constructors

HttpClient.CommandEventArgs Constructor	Initializes a new instance of the HttpClient.CommandEventArgs class.

Public Instance Properties

ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpClient.CommandEventArgs Class | SocketTools Namespace

HttpClient.CommandEventArgs Constructor

Initializes a new instance of the HttpClient.CommandEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public HttpClient.CommandEventArgs();
```

See Also

HttpClient.CommandEventArgs Class | SocketTools Namespace

HttpClient.CommandEventArgs Properties

The properties of the **HttpClient.CommandEventArgs** class are listed below. For a complete list of **HttpClient.CommandEventArgs** class members, see the HttpClient.CommandEventArgs Members topic.

Public Instance Properties

ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.

See Also

HttpClient.CommandEventArgs Class | SocketTools Namespace

HttpClient.CommandEventArgs.ResultCode Property

Gets a value which specifies the last result code returned by the server.

[Visual Basic]
```
Public ReadOnly Property ResultCode As Integer
```

[C#]
```
public int ResultCode {get;}
```

Property Value

An integer value which specifies the last result code returned by the server.

Remarks

This property should be checked after the **Command** method is used to execute a command on the server to determine if the operation was successful. Result codes are three-digit numeric values returned by the remote server and may be broken down into the following ranges:

ResultCode	Description
100-199	Positive preliminary result. This indicates that the requested action is being initiated, and the client should expect another reply from the server before proceeding.
200-299	Positive completion result. This indicates that the server has successfully completed the requested action.
300-399	Positive intermediate result. This indicates that the requested action cannot complete until additional information is provided to the server.
400-499	Transient negative completion result. This indicates that the requested action did not take place, but the error condition is temporary and may be attempted again.
500-599	Permanent negative completion result. This indicates that the requested action did not take place.

It is important to note that while some result codes have become standardized, not all servers respond to commands using the same result codes. For example, one server may respond with a result code of 221 to indicate success, while another may respond with a value of 235. It is recommended that applications check for ranges of values to determine if a command was successful, not a specific value.

See Also

HttpClient.CommandEventArgs Class | SocketTools Namespace

HttpClient.CommandEventArgs.ResultString Property

Gets a string value which describes the result of the previous command.

[Visual Basic]

```
Public ReadOnly Property ResultString As String
```

[C#]

```
public string ResultString {get;}
```

Property Value

A string which describes the result of the previous command executed on the server.

Remarks

This string is generated by the remote server, and typically is used to describe the result code. For example, if an error is indicated by the result code, the result string may describe the condition that caused the error.

See Also

HttpClient.CommandEventArgs Class | SocketTools Namespace

HttpClient.OnConnect Event

Occurs when a connection is established with the remote host.

[Visual Basic]

```
Public Event OnConnect As EventHandler
```

[C#]

```
public event EventHandler OnConnect;
```

Remarks

The **OnConnect** event occurs when a connection is made with a remote host as a result of a **Connect** method call. When the **Connect** method is called and the **Blocking** property is set to **false**, a socket is created but the connection is not actually established until after this event occurs. Between the time connection process is started and this event fires, no operation may be performed on the client other than calling the **Disconnect** method.

This event is only generated if the client is in non-blocking mode.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.OnDisconnect Event

Occurs when the remote host disconnects from the local system.

[Visual Basic]

```
Public Event OnDisconnect As EventHandler
```

[C#]

```
public event EventHandler OnDisconnect;
```

Remarks

The **OnDisconnect** event occurs when the remote host closes its connection, terminating the client session with the application. Because there may still be data in the client receive buffers, you should continue to read data from the client until the **Read** method returns a value of 0. Once all of the data has been read, you should call the **Disconnect** method to close the local socket and release the resources allocated for the client.

This event is only generated if the client is in non-blocking mode.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.OnError Event

Occurs when an client operation fails.

[Visual Basic]

```
Public Event OnError As OnErrorEventHandler
```

[C#]

```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type HttpClient.ErrorEventArgs containing data related to this event. The following **HttpClient.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a client operation fails.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see HttpClient.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.HttpClient.ErrorEventArgs

[Visual Basic]

```
Public Class HttpClient.ErrorEventArgs
    Inherits EventArgs
```

[C#]

```
public class HttpClient.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

HttpClient.ErrorEventArgs Members | SocketTools Namespace

HttpClient.ErrorEventArgs Members

HttpClient.ErrorEventArgs overview

Public Instance Constructors

HttpClient.ErrorEventArgs Constructor	Initializes a new instance of the HttpClient.ErrorEventArgs class.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpClient.ErrorEventArgs Class | SocketTools Namespace

HttpClient.ErrorEventArgs Constructor

Initializes a new instance of the HttpClient.ErrorEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public HttpClient.ErrorEventArgs();
```

See Also

HttpClient.ErrorEventArgs Class | SocketTools Namespace

HttpClient.ErrorEventArgs Properties

The properties of the **HttpClient.ErrorEventArgs** class are listed below. For a complete list of **HttpClient.ErrorEventArgs** class members, see the HttpClient.ErrorEventArgs Members topic.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

HttpClient.ErrorEventArgs Class | SocketTools Namespace

HttpClient.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

HttpClient.ErrorEventArgs Class | SocketTools Namespace | Error Property

HttpClient.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public HttpClient.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

HttpClient.ErrorEventArgs Class | SocketTools Namespace | Description Property

HttpClient.OnProgress Event

Occurs as a data stream is being read or written to the client.

[Visual Basic]
```
Public Event OnProgress As OnProgressEventHandler
```

[C#]
```
public event OnProgressEventHandler OnProgress;
```

Event Data

The event handler receives an argument of type HttpClient.ProgressEventArgs containing data related to this event. The following **HttpClient.ProgressEventArgs** properties provide information specific to this event.

Property	Description
BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
BytesTotal	Gets a value which specifies the total number of bytes in the data stream.
Percent	Gets a value which specifies the percentage of data that has been read or written.
Resource	Gets a value which specifies the resource or file name.

Remarks

The **OnProgress** event occurs as a data stream is being read or written to the client. If large amounts of data are being read or written, this event can be used to update a progress bar or other user-interface component to provide the user with some visual feedback on the progress of the operation.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.ProgressEventArgs Class

Provides data for the OnProgress event.

For a list of all members of this type, see HttpClient.ProgressEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.HttpClient.ProgressEventArgs

[Visual Basic]
```
Public Class HttpClient.ProgressEventArgs
    Inherits EventArgs
```

[C#]
```
public class HttpClient.ProgressEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ProgressEventArgs specifies the number of bytes copied from the data stream, the total number of bytes in the data stream and a completion percentage.

The OnProgress event occurs as a data stream is being read or written to the client. This event only occurs when the **GetData, GetFile**, **PostData, PostFile, PutData, PutFile** or **SubmitForm** methods are called.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

HttpClient.ProgressEventArgs Members | SocketTools Namespace

HttpClient.ProgressEventArgs Members

HttpClient.ProgressEventArgs overview

Public Instance Constructors

HttpClient.ProgressEventArgs Constructor	Initializes a new instance of the HttpClient.ProgressEventArgs class.

Public Instance Properties

BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
BytesTotal	Gets a value which specifies the total number of bytes in the data stream.
Percent	Gets a value which specifies the percentage of data that has been read or written.
Resource	Gets a value which specifies the resource or file name.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpClient.ProgressEventArgs Class | SocketTools Namespace

HttpClient.ProgressEventArgs Constructor

Initializes a new instance of the HttpClient.ProgressEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public HttpClient.ProgressEventArgs();
```

See Also

HttpClient.ProgressEventArgs Class | SocketTools Namespace

HttpClient.ProgressEventArgs Properties

The properties of the **HttpClient.ProgressEventArgs** class are listed below. For a complete list of **HttpClient.ProgressEventArgs** class members, see the HttpClient.ProgressEventArgs Members topic.

Public Instance Properties

BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
BytesTotal	Gets a value which specifies the total number of bytes in the data stream.
Percent	Gets a value which specifies the percentage of data that has been read or written.
Resource	Gets a value which specifies the resource or file name.

See Also

HttpClient.ProgressEventArgs Class | SocketTools Namespace

HttpClient.ProgressEventArgs.BytesCopied Property

Gets a value which specifies the number of bytes of data that has been read or written.

[Visual Basic]

```
Public ReadOnly Property BytesCopied As Long
```

[C#]

```
public long BytesCopied {get;}
```

Property Value

An integer value which specifies the number of bytes of data.

Remarks

The **BytesCopied** property specifies the number of bytes that have been read from the client and stored in the local stream buffer, or written from the stream buffer to the client.

See Also

HttpClient.ProgressEventArgs Class | SocketTools Namespace | BytesTotal Property | Percent Property

HttpClient.ProgressEventArgs.BytesTotal Property

Gets a value which specifies the total number of bytes in the data stream.

[Visual Basic]

```
Public ReadOnly Property BytesTotal As Long
```

[C#]

```
public long BytesTotal {get;}
```

Property Value

An integer value which specifies the number of bytes of data.

Remarks

The **BytesTotal** property specifies the total amount of data being read from the client and stored in the data stream, or written from the data stream to the client. If the amount of data was unknown or unspecified at the time the method call was made, then this value will always be the same as the **BytesCopied** property.

See Also

HttpClient.ProgressEventArgs Class | SocketTools Namespace | BytesCopied Property | Percent Property

HttpClient.ProgressEventArgs.Percent Property

Gets a value which specifies the percentage of data that has been read or written.

[Visual Basic]

```
Public ReadOnly Property Percent As Integer
```

[C#]

```
public int Percent {get;}
```

Property Value

An integer value which specifies a percentage.

Remarks

The **Percent** property specifies the percentage of data that has been transmitted, expressed as an integer value between 0 and 100, inclusive. If the maximum size of the data stream was not specified by the caller, this value will always be 100.

See Also

HttpClient.ProgressEventArgs Class | SocketTools Namespace | BytesCopied Property | BytesTotal Property

HttpClient.OnRead Event

Occurs when data is available to be read from the client.

[Visual Basic]

```
Public Event OnRead As EventHandler
```

[C#]

```
public event EventHandler OnRead;
```

Remarks

The **OnRead** event occurs when data is available to be read from the client. This event is level-triggered, which means that once this event fires, it will not occur again until some data has been read from the client. This design prevents an application from being flooded with event notifications. It is recommended that your application read all of the available data from the server and store it in a local buffer for processing. See the example below.

This event is only generated if the client is in non-blocking mode.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.OnRedirect Event

Occurs when the server indicates a resource has been moved.

[Visual Basic]
```
Public Event OnRedirect As OnRedirectEventHandler
```

[C#]
```
public event OnRedirectEventHandler OnRedirect;
```

Event Data

The event handler receives an argument of type HttpClient.RedirectEventArgs containing data related to this event. The following **HttpClient.RedirectEventArgs** property provides information specific to this event.

Property	Description
Location	Gets the location of the redirected resource.

Remarks

The **OnRedirect** event is generated when the server indicates that the requested resource has been moved to a new location. This new location is typically on the same server, however it may specify another server.

If the **AutoRedirect** property is set to **true**, then the class will automatically retrieve the resource from its new location. If the property is set to **false**, then the application is responsible for handling the redirection.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.RedirectEventArgs Class

Provides data for the OnRedirect event.

For a list of all members of this type, see HttpClient.RedirectEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.HttpClient.RedirectEventArgs

[Visual Basic]

```
Public Class HttpClient.RedirectEventArgs
    Inherits EventArgs
```

[C#]

```
public class HttpClient.RedirectEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

RedirectEventArgs provides the location of the redirected resource.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

HttpClient.RedirectEventArgs Members | SocketTools Namespace

HttpClient.RedirectEventArgs Members

HttpClient.RedirectEventArgs overview

Public Instance Constructors

HttpClient.RedirectEventArgs Constructor	Initializes a new instance of the HttpClient.RedirectEventArgs class.

Public Instance Properties

Location	Gets the location of the redirected resource.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpClient.RedirectEventArgs Class | SocketTools Namespace

HttpClient.RedirectEventArgs Constructor

Initializes a new instance of the HttpClient.RedirectEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public HttpClient.RedirectEventArgs();
```

See Also

HttpClient.RedirectEventArgs Class | SocketTools Namespace

HttpClient.RedirectEventArgs Properties

The properties of the **HttpClient.RedirectEventArgs** class are listed below. For a complete list of **HttpClient.RedirectEventArgs** class members, see the HttpClient.RedirectEventArgs Members topic.

Public Instance Properties

Location	Gets the location of the redirected resource.

See Also

HttpClient.RedirectEventArgs Class | SocketTools Namespace

HttpClient.RedirectEventArgs.Location Property

Gets the location of the redirected resource.

[Visual Basic]

```
Public ReadOnly Property Location As String
```

[C#]

```
public string Location {get;}
```

Property Value

A string which specifies the new location for the resource.

See Also

HttpClient.RedirectEventArgs Class | SocketTools Namespace

HttpClient.OnTaskBegin Event

Occurs when an asynchronous task begins execution.

[Visual Basic]

```
Public Event OnTaskBegin As OnTaskBeginEventHandler
```

[C#]

```
public event OnTaskBeginEventHandler OnTaskBegin;
```

Event Data

The event handler receives an argument of type HttpClient.TaskBeginEventArgs containing data related to this event. The following **HttpClient.TaskBeginEventArgs** property provides information specific to this event.

Property	Description
TaskId	Get the unique task identifier associated with the event.

Remarks

The **OnTaskBegin** event occurs when a background task associated with an asynchronous file transfer begins executing. This event can be used in conjunction with the **OnTaskEnd** event to monitor one or more background tasks that are created to perform asynchronous file transfers.

This event and the related asynchronous task events are invoked from the context of the thread that is managing the background task, and not the thread that created the class instance. If a handler is implemented for this event, its code will be executing in a different thread than the main UI thread. You should never attempt to update your application's user interface directly from within this event handler. Instead, you must create a delegate and use the **Invoke** method to ensure that any changes to the user interface are done within the context of the main UI thread.

Because background tasks are managed in separate threads, this has the effect of making your application multi-threaded, even if you do not explicitly create any worker threads in your own code. If the code in your event handler modifies a public member variable or shared object, you must ensure that access to that object is synchronized. For example, if your event handler updates a shared instance of a **Hashtable** object, you should ensure that all operations are performed through the thread-safe wrapper returned by the **Synchronized** method for that class. Refer to the MSDN documentation for more information about creating thread-safe applications.

See Also

HttpClient Class | SocketTools Namespace | OnTaskEnd Event | OnTaskRun Event

HttpClient.TaskBeginEventArgs Class

Provides data for the OnTaskBegin event.

For a list of all members of this type, see HttpClient.TaskBeginEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.HttpClient.TaskBeginEventArgs

[Visual Basic]

```
Public Class HttpClient.TaskBeginEventArgs
    Inherits EventArgs
```

[C#]

```
public class HttpClient.TaskBeginEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

HttpClient.TaskBeginEventArgs Members | SocketTools Namespace

HttpClient.TaskBeginEventArgs Constructor

Initializes a new instance of the HttpClient.TaskBeginEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public HttpClient.TaskBeginEventArgs();
```

See Also

HttpClient.TaskBeginEventArgs Class | SocketTools Namespace

HttpClient.TaskBeginEventArgs Members

HttpClient.TaskBeginEventArgs overview

Public Instance Constructors

HttpClient.TaskBeginEventArgs Constructor	Initializes a new instance of the HttpClient.TaskBeginEventArgs class.

Public Instance Properties

TaskId	Get the unique task identifier associated with the event.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpClient.TaskBeginEventArgs Class | SocketTools Namespace

HttpClient.TaskBeginEventArgs Properties

The properties of the **HttpClient.TaskBeginEventArgs** class are listed below. For a complete list of **HttpClient.TaskBeginEventArgs** class members, see the HttpClient.TaskBeginEventArgs Members topic.

Public Instance Properties

TaskId	Get the unique task identifier associated with the event.

See Also

HttpClient.TaskBeginEventArgs Class | SocketTools Namespace

HttpClient.TaskBeginEventArgs.TaskId Property

Get the unique task identifier associated with the event.

[Visual Basic]

```
Public ReadOnly Property TaskId As Integer
```

[C#]

```
public int TaskId {get;}
```

Property Value

An integer value that uniquely identifies the task that invoked the event handler.

See Also

HttpClient.TaskBeginEventArgs Class | SocketTools Namespace

HttpClient.OnTaskEnd Event

Occurs when an asynchronous task completes.

[Visual Basic]
```
Public Event OnTaskEnd As OnTaskEndEventHandler
```

[C#]
```
public event OnTaskEndEventHandler OnTaskEnd;
```

Event Data

The event handler receives an argument of type HttpClient.TaskEndEventArgs containing data related to this event. The following **HttpClient.TaskEndEventArgs** properties provide information specific to this event.

Property	Description
Error	Get the last error code for the background task.
TaskId	Get the unique task identifier associated with the event.
TimeElapsed	Gets the amount of time that has elapsed in milliseconds.

Remarks

The **OnTaskEnd** event occurs when a file transfer completes and the background task has terminated. Refer to the **OnTaskBegin** event for additional information about implementing a handler for this event.

See Also

HttpClient Class | SocketTools Namespace | OnTaskBegin Event | OnTaskRun Event

HttpClient.TaskEndEventArgs Class

Provides data for the OnTaskEnd event.

For a list of all members of this type, see HttpClient.TaskEndEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.HttpClient.TaskEndEventArgs

[Visual Basic]

```
Public Class HttpClient.TaskEndEventArgs
    Inherits EventArgs
```

[C#]

```
public class HttpClient.TaskEndEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

HttpClient.TaskEndEventArgs Members | SocketTools Namespace

HttpClient.TaskEndEventArgs Constructor

Initializes a new instance of the HttpClient.TaskEndEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public HttpClient.TaskEndEventArgs();
```

See Also

HttpClient.TaskEndEventArgs Class | SocketTools Namespace

HttpClient.TaskEndEventArgs Members

HttpClient.TaskEndEventArgs overview

Public Instance Constructors

HttpClient.TaskEndEventArgs Constructor	Initializes a new instance of the HttpClient.TaskEndEventArgs class.

Public Instance Properties

Error	Get the last error code for the background task.
TaskId	Get the unique task identifier associated with the event.
TimeElapsed	Gets the amount of time that has elapsed in milliseconds.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpClient.TaskEndEventArgs Class | SocketTools Namespace

HttpClient.TaskEndEventArgs Properties

The properties of the **HttpClient.TaskEndEventArgs** class are listed below. For a complete list of **HttpClient.TaskEndEventArgs** class members, see the HttpClient.TaskEndEventArgs Members topic.

Public Instance Properties

Error	Get the last error code for the background task.
TaskId	Get the unique task identifier associated with the event.
TimeElapsed	Gets the amount of time that has elapsed in milliseconds.

See Also

HttpClient.TaskEndEventArgs Class | SocketTools Namespace

HttpClient.TaskEndEventArgs.Error Property

Get the last error code for the background task.

[Visual Basic]
```
Public ReadOnly Property Error As ErrorCode
```

[C#]
```
public HttpClient.ErrorCode Error {get;}
```

Property Value

An **ErrorCode** enumeration that specifies the last error code set by the background task.

See Also

HttpClient.TaskEndEventArgs Class | SocketTools Namespace

HttpClient.TaskEndEventArgs.TaskId Property

Get the unique task identifier associated with the event.

[Visual Basic]

```
Public ReadOnly Property TaskId As Integer
```

[C#]

```
public int TaskId {get;}
```

Property Value

An integer value that uniquely identifies the task that invoked the event handler.

See Also

HttpClient.TaskEndEventArgs Class | SocketTools Namespace

HttpClient.TaskEndEventArgs.TimeElapsed Property

Gets the amount of time that has elapsed in milliseconds.

[Visual Basic]

```
Public ReadOnly Property TimeElapsed As Integer
```

[C#]

```
public int TimeElapsed {get;}
```

Property Value

An integer value that specifies the number of milliseconds that the background task has executed.

See Also

HttpClient.TaskEndEventArgs Class | SocketTools Namespace

HttpClient.OnTaskRun Event

Occurs while a background task is active.

[Visual Basic]
```
Public Event OnTaskRun As OnTaskRunEventHandler
```

[C#]
```
public event OnTaskRunEventHandler OnTaskRun;
```

Event Data

The event handler receives an argument of type HttpClient.TaskRunEventArgs containing data related to this event. The following **HttpClient.TaskRunEventArgs** properties provide information specific to this event.

Property	Description
Completed	Gets an estimate of the progress of the background task.
TaskId	Get the unique task identifier associated with the event.
TimeElapsed	Gets the amount of time that has elapsed in milliseconds.

Remarks

The **OnTaskRun** event is generated periodically during a file transfer while the background task is active. The rate and number of times that this event will be generated depends on the task being performed. This event is generally analogous to the **OnProgress** event for file transfers that are performed in the current working thread, however the **OnTaskRun** event will occur for each individual background task that is active.

Refer to the **OnTaskBegin** event for additional information about implementing a handler for this event.

See Also

HttpClient Class | SocketTools Namespace | OnTaskBegin Event | OnTaskEnd Event

HttpClient.TaskRunEventArgs Class

Provides data for the OnTaskRun event.

For a list of all members of this type, see HttpClient.TaskRunEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.HttpClient.TaskRunEventArgs

[Visual Basic]

```
Public Class HttpClient.TaskRunEventArgs
    Inherits EventArgs
```

[C#]

```
public class HttpClient.TaskRunEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

HttpClient.TaskRunEventArgs Members | SocketTools Namespace

HttpClient.TaskRunEventArgs Constructor

Initializes a new instance of the HttpClient.TaskRunEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public HttpClient.TaskRunEventArgs();
```

See Also

HttpClient.TaskRunEventArgs Class | SocketTools Namespace

HttpClient.TaskRunEventArgs Members

HttpClient.TaskRunEventArgs overview

Public Instance Constructors

HttpClient.TaskRunEventArgs Constructor	Initializes a new instance of the HttpClient.TaskRunEventArgs class.

Public Instance Properties

Completed	Gets an estimate of the progress of the background task.
TaskId	Get the unique task identifier associated with the event.
TimeElapsed	Gets the amount of time that has elapsed in milliseconds.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpClient.TaskRunEventArgs Class | SocketTools Namespace

HttpClient.TaskRunEventArgs Properties

The properties of the **HttpClient.TaskRunEventArgs** class are listed below. For a complete list of **HttpClient.TaskRunEventArgs** class members, see the HttpClient.TaskRunEventArgs Members topic.

Public Instance Properties

Completed	Gets an estimate of the progress of the background task.
TaskId	Get the unique task identifier associated with the event.
TimeElapsed	Gets the amount of time that has elapsed in milliseconds.

See Also

HttpClient.TaskRunEventArgs Class | SocketTools Namespace

HttpClient.TaskRunEventArgs.Completed Property

Gets an estimate of the progress of the background task.

[Visual Basic]

```
Public ReadOnly Property Completed As Integer
```

[C#]

```
public int Completed {get;}
```

Property Value

An integer value that returns a number between 0 and 100 inclusive that specifies the estimated percentage of completion for the task. A value of zero indicates that the task has just begun executing, while a value of 100 indicates that the task is at or near completion.

See Also

HttpClient.TaskRunEventArgs Class | SocketTools Namespace

HttpClient.TaskRunEventArgs.TaskId Property

Get the unique task identifier associated with the event.

[Visual Basic]

```
Public ReadOnly Property TaskId As Integer
```

[C#]

```
public int TaskId {get;}
```

Property Value

An integer value that uniquely identifies the task that invoked the event handler.

See Also

HttpClient.TaskRunEventArgs Class | SocketTools Namespace

HttpClient.TaskRunEventArgs.TimeElapsed Property

Gets the amount of time that has elapsed in milliseconds.

[Visual Basic]

```
Public ReadOnly Property TimeElapsed As Integer
```

[C#]

```
public int TimeElapsed {get;}
```

Property Value

An integer value that specifies the number of milliseconds that the background task has been executing.

See Also

HttpClient.TaskRunEventArgs Class | SocketTools Namespace

HttpClient.OnTimeout Event

Occurs when a blocking operation fails to complete before the timeout period elapses.

[Visual Basic]

```
Public Event OnTimeout As EventHandler
```

[C#]

```
public event EventHandler OnTimeout;
```

Remarks

The **OnTimeout** event occurs when a blocking operation, such as sending or receiving data on the client, fails to complete before the specified timeout period elapses. The timeout period for a blocking operation can be adjusted by setting the **Timeout** property.

This event is only generated if the client is in blocking mode.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.OnWrite Event

Occurs when data can be written to the client.

[Visual Basic]

```
Public Event OnWrite As EventHandler
```

[C#]

```
public event EventHandler OnWrite;
```

Remarks

The **OnWrite** event occurs when the application can write data to the client. This event will typically occur when a connection is first established with the remote host, and after the **Write** method has failed because there was insufficient memory available in the client send buffers. In the second case, when some of the buffered data has been successfully sent to the remote host and there is space available in the send buffers, this event is used to signal the application that it may attempt to send more data.

This event is only generated if the client is in non-blocking mode.

See Also

HttpClient Class | SocketTools Namespace

HttpClient.CookieFlags Enumeration

Specifies the cookie flags supported by the HttpClient class.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum HttpClient.CookieFlags
```

[C#]

```
[Flags]
public enum HttpClient.CookieFlags
```

Remarks

The HttpClient class uses the **CookieFlags** enumeration to specify one or more cookie flags when a cookie is set by the server. Multiple options may be specified.

Members

Member Name	Description	Value
cookieDefault	This flag specifies that the cookie can be stored on the local system and may be provided to the server over a standard, non-secure connection.	0
cookieSecure	This flag specifies that the cookie should only be provided to the server if the connection is secure.	1
cookieSession	This flag specifies that the cookie should only be used for the current application session and should not be stored permanently on the local system.	2

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

SocketTools Namespace

HttpClient.ErrorCode Enumeration

Specifies the error codes returned by the HttpClient class.

[Visual Basic]

```
Public Enum HttpClient.ErrorCode
```

[C#]

```
public enum HttpClient.ErrorCode
```

Remarks

The HttpClient class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.
errorProxyAuthenticationRequired	Proxy authentication required.

errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.
errorUnsupportedMediaType	Request specified an unsupported media type.

errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.
errorNoMoreMailboxes	No more mailboxes exist on this server.

errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.
errorInvalidThreadHandle	The specified thread handle is no longer valid.

errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been defined.

errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or unrecognized.

errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

HttpClient.HttpFormMethod Enumeration

Specifies the methods supported by the HttpClient class when submitting form data to a server.

[Visual Basic]

```
Public Enum HttpClient.HttpFormMethod
```

[C#]

```
public enum HttpClient.HttpFormMethod
```

Remarks

The HttpClient class uses the **HttpFormMethod** enumeration to specify the method that should be used when submitting form data to a server for processing.

Members

Member Name	Description
methodDefault	The form data should be submitted using the default method, using the GET command.
methodGet	The form data should be submitted using the GET command. This method should be used when the amount of form data is relatively small. If the total amount of form data exceeds 2048 bytes, it is recommended that the POST method be used instead.
methodPost	The form data should be submitted using the POST command. This is the preferred method of submitting larger amounts of form data. If the total amount of form data exceeds 2048 bytes, it is recommended that the POST method be used.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

SocketTools Namespace

HttpClient.HttpEncoding Enumeration

Specifies the encoding types supported by the HttpClient class.

[Visual Basic]

```
Public Enum HttpClient.HttpEncoding
```

[C#]

```
public enum HttpClient.HttpEncoding
```

Remarks

The HttpClient class uses the **HttpEncoding** enumeration to specify the type of encoding to be used when data is submitted to the server for processing.

Members

Member Name	Description
encodingNone	No encoding will be applied to the content of a request, and no Content-Type header will be generated.
encodingURL	Standard URL encoding will be applied to the content of a request, and a Content-Type header will be generated with the value "application/x-www-form-urlencoded". This is the default encoding type.
encodingXML	Standard URL encoding will be applied to the content of a request, except that spaces will not be replaced by '+'. This encoding type is intended for use with XML parsers that do not recognize '+' as a space. A Content-Type header will be generated with the value "application/x-www-form-urlencoded".

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

SocketTools Namespace

HttpClient.HttpFormType Enumeration

Specifies the form types supported by the HttpClient class when submitting form data to a server.

[Visual Basic]

```
Public Enum HttpClient.HttpFormType
```

[C#]

```
public enum HttpClient.HttpFormType
```

Remarks

The HttpClient class uses the **HttpFormType** enumeration to specify how form data that should be submitted to a server for processing.

Members

Member Name	Description
formDefault	The form data should be submitted using the default encoding method.
formEncoded	The form data should be submitted as URL encoded values. This is typically used when the GET method is used to submit the data to the server.
formMultipart	The form data should be submitted as multipart form data. This is typically used when the POST method is used to submit a file to the server. Note that the script must understand how to process multipart form data if this form type is specified.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

SocketTools Namespace

HttpClient.HttpOptions Enumeration

Specifies the options that the HttpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum HttpClient.HttpOptions
```

[C#]
```
[Flags]
public enum HttpClient.HttpOptions
```

Remarks

The HttpClient class uses the **HttpOptions** enumeration to specify one or more options to be used when establishing a connection with a remote host. Multiple options may be specified if necessary.

Members

Member Name	Description	Value
optionNone	No option specified.	0
optionNoCache	This instructs the server to not return a cached copy of the resource. When connected to an HTTP 1.0 or earlier server, this directive may be ignored.	1
optionKeepAlive	This instructs the server to maintain a persistent connection between requests. This can improve performance because it eliminates the need to establish a separate connection for each resource that is requested. If the server does not support the keep-alive option, the client will automatically reconnect when each resource is requested. Although it will not provide any performance benefits, this allows the option to be used with all servers.	2
optionRedirect	This option specifies the client should automatically handle resource redirection. If the server indicates that the requested resource has moved to a new location, the client will close the current connection and request the resource from the new location. Note that it is possible that the redirected resource will be located on a different server.	4
optionProxy	This option specifies the client should use the default proxy configuration for	8

	the local system. If the system is configured to use a proxy server, then the connection will be automatically established through that proxy; otherwise, a direct connection to the server is established. The local proxy configuration can be changed using the system Control Panel.	
optionErrorData	This option specifies the client should return the content of an error response from the server, rather than returning an error code. Note that this option will disable automatic resource redirection, and should not be used with optionRedirect.	16
optionTunnel	This option specifies that a tunneled TCP connection and/or port-forwarding is being used to establish the connection to the server. This changes the behavior of the client with regards to internal checks of the destination IP address and remote port number, default capability selection and how the connection is established. This option also forces all connections to be outbound and enables the firewall compatibility features in the client.	1024
optionTrustedSite	This option specifies the server is trusted. The server certificate will not be validated and the connection will always be permitted. This option only affects connections using either the SSL or TLS protocols.	2048
optionSecure	This option specifies the client should attempt to establish a secure connection with the server. Note that the server must support secure connections using either the SSL or TLS protocol.	4096
optionSecureFallback	This option specifies the client should permit the use of less secure cipher suites for compatibility with legacy servers. If this option is specified, the client will permit connections using TLS 1.0 and cipher suites that use RC4, MD5 and SHA1.	32768
optionPreferIPv6	This option specifies the client should prefer the use of IPv6 if the server hostname can be resolved to both an	262144

	IPv6 and IPv4 address. This option is ignored if the local system does not have IPv6 enabled, or when the hostname can only be resolved to an IPv4 address. If the server hostname can only be resolved to an IPv6 address, the client will attempt to establish a connection using IPv6 regardless if this option has been specified.	
optionFreeThread	This option specifies that class methods may be called from any thread, and not only the thread that established the connection. Using this option disables certain internal safety checks that are made by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads.	524288
optionHiResTimer	This option specifies that elapsed time values should be returned in milliseconds rather than seconds. This option is intended to provide greater accuracy with smaller data transfers over a high speed network connection.	1048576

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

SocketTools Namespace

HttpClient.HttpPostOptions Enumeration

Specifies options that the HttpClient class supports when posting data to a server.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum HttpClient.HttpPostOptions
```

[C#]

```
[Flags]
public enum HttpClient.HttpPostOptions
```

Remarks

The HttpClient class uses the **HttpPostOptions** enumeration to specify one or more options to be used when posting data to the server. Multiple options may be specified if necessary.

Members

Member Name	Description	Value
postDefault	The default post mode with appropriate encoding used for the type of data which is being submitted to the server. The response from the server will be returned in the result buffer without any end-of-line text conversion.	0
postConvert	If the data being returned from the server is textual, it is automatically converted so that the end of line character sequence is compatible with the Windows platform. Individual carriage return or linefeed characters are converted to carriage return/linefeed character sequences. Note that this option does not have any effect on the data being submitted to the server, only on the data returned by the server.	1
postMultipart	The contents of the buffer being sent to the server consists of multipart form data and will be sent as-is without any encoding.	2
postErrorData	This option causes the client to accept error data from the server if the request fails. If this option is specified, an error response from the server will not cause the method to fail. Instead, the response is returned to the client and the method will succeed. If this option is used, your application should check the value of	4

	the **ResultCode** property to obtain the actual HTTP status code returned by the server. This will enable you to determine if the operation was successful.	

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

SocketTools Namespace

HttpClient.HttpProxyType Enumeration

Specifies the proxy server types supported by the HttpClient class.

[Visual Basic]
```
Public Enum HttpClient.HttpProxyType
```

[C#]
```
public enum HttpClient.HttpProxyType
```

Remarks

The HttpClient class uses the **HttpProxyType** enumeration to specify the type of proxy server the client is connecting to.

Members

Member Name	Description
proxyNone	A direct connection will be established with the remote host. When this value is specified the proxy-related properties are ignored.
proxyStandard	A standard connection is established through the specified proxy server, and all resource requests will be specified using a complete URL. This proxy type should be used with standard connections.
proxySecure	A secure connection is established through the specified proxy server. This proxy type should only be used with secure connections and the Secure property should also be set to a value of true.
proxyWindows	The configuration options for the current system should be used. These options are found in the Network and Internet settings for Windows.
proxyDefault	The default proxy type. This value is the same as the **proxyWindows** proxy type.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

SocketTools Namespace

HttpClient.HttpStatus Enumeration

Specifies the status values that may be returned by the HttpClient class.

[Visual Basic]

```
Public Enum HttpClient.HttpStatus
```

[C#]

```
public enum HttpClient.HttpStatus
```

Remarks

The HttpClient class uses the **HttpStatus** enumeration to identify the current status of the client.

Members

Member Name	Description
statusUnused	A client session has not been created. Attempts to perform any network operations, such as sending or receiving data, will generate an error.
statusIdle	A client session has been created, but is not currently in use. A blocking socket operation can be executed at this point.
statusConnect	The client is in the process of establishing a connection with a remote host.
statusRead	The client is in the process of receiving data from a remote host.
statusWrite	The client is in the process of sending data to a remote host.
statusDisconnect	The client session is being closed and subsequent attempts to access the client will result in an error.
statusGetData	The client is downloading data from the server to the local system.
statusPutData	The client is uploading data from the local system to the server.
statusPostData	The client is posting form data to the server.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

SocketTools Namespace

HttpClient.HttpSubmitOptions Enumeration

Specifies the options that the HttpClient class supports when submitting form data to a server.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum HttpClient.HttpSubmitOptions
```

[C#]

```
[Flags]
public enum HttpClient.HttpSubmitOptions
```

Remarks

The HttpClient class uses the **HttpSubmitOptions** enumeration to specify one or more options to be used when submitting form data to a server for processing. Multiple options may be specified if necessary.

Members

Member Name	Description	Value
submitDefault	The default post mode. The contents of the buffer are encoded and sent as standard form data. The response from the server will be returned in the result buffer without any end-of-line text conversion.	0
submitConvert	If the data being returned from the server is textual, it is automatically converted so that the end of line character sequence is compatible with the Windows platform. Individual carriage return or linefeed characters are converted to carriage return/linefeed character sequences. Note that this option does not have any effect on the form data being submitted to the server, only on the data returned by the server.	1

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

SocketTools Namespace

HttpClient.HttpTransferOptions Enumeration

Specifies the data transfer options that the HttpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum HttpClient.HttpTransferOptions
```

[C#]

```
[Flags]
public enum HttpClient.HttpTransferOptions
```

Remarks

The HttpClient class uses the **HttpTransferOptions** enumeration to specify one or more options to be used when transferring data between the local system and the server. Multiple options may be specified if necessary.

Members

Member Name	Description	Value
transferDefault	The default transfer mode. The resource data is copied to the local system exactly as it is stored on the server.	0
transferConvert	If the resource being downloaded from the server is textual, the data is automatically converted so that the end of line character sequence is compatible with the Windows platform. Individual carriage return or linefeed characters are converted to carriage return/linefeed character sequences.	1
transferCompress	This option informs the server that the client is willing to accept compressed data. If the server supports compression for the specified resource, then the data will be automatically expanded before being returned to the caller. This option is selected by default if compression has been enabled by setting the Compression property to True.	2
transferErrorData	This option causes the client to accept error data from the server if the request fails. If this option is specified, an error response from the server will not cause the method to fail. Instead, the response is returned to the client and the method will succeed. If this option is used, your application should check the value of the **ResultCode** property to obtain the	4

	actual HTTP status code returned by the server. This will enable you to determine if the operation was successful.	

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

SocketTools Namespace

HttpClient.HttpVersion Enumeration

Specifies the protocol versions supported by the HttpClient class.

[Visual Basic]

```
Public Enum HttpClient.HttpVersion
```

[C#]

```
public enum HttpClient.HttpVersion
```

Remarks

The HttpClient class uses the **HttpVersion** enumeration to specify the protocol version to be used when establishing a connection to the server.

Members

Member Name	Description
version09	Version 0.9 of the protocol. This value specifies that the client should use the preliminary protocol version which only supports the use of the GET command. Header fields are not supported with this version of the protocol.
version10	Version 1.0 of the protocol.
version11	Version 1.1 of the protocol.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

SocketTools Namespace

HttpClient.SecureCipherAlgorithm Enumeration

Specifies the encryption algorithms that the HttpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum HttpClient.SecureCipherAlgorithm
```

[C#]
```
[Flags]
public enum HttpClient.SecureCipherAlgorithm
```

Remarks

The HttpClient class uses the **SecureCipherAlgorithm** enumeration to identify which encryption algorithm was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
cipherNone	No cipher has been selected. A secure connection has not been established with the remote host.	0
cipherRC2	The RC2 block cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.	1
cipherRC4	The RC4 stream cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.	2
cipherRC5	The RC5 block cipher was selected. This is a variable key length cipher which supports keys up to 2040 bits, in 8-bit increments.	4
cipherDES	The DES (Data Encryption Standard) block cipher was selected. This is a fixed key length cipher using 56-bit keys.	8
cipherDES3	The Triple DES block cipher was selected. This cipher encrypts the data three times using different keys, effectively using a 168-bit key length.	16
cipherDESX	A variant of the DES block cipher which XORs an extra 64-bits of the key before and after the plaintext has been encrypted, increasing the key size to 184 bits.	32
cipherAES	The Advanced Encryption Standard	64

	cipher (also known as the Rijndael cipher) is a fixed block size cipher which use a key size of 128, 192 or 256 bits. This cipher is supported on Windows XP SP3 SP3 and later versions of the operating system.	
cipherSkipjack	The Skipjack block cipher was selected. This is a fixed key length cipher, using 80-bit keys.	128

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

SocketTools Namespace

HttpClient.SecureHashAlgorithm Enumeration

Specifies the hash algorithms that the HttpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum HttpClient.SecureHashAlgorithm
```

[C#]
```
[Flags]
public enum HttpClient.SecureHashAlgorithm
```

Remarks

The HttpClient class uses the **SecureHashAlgorithm** enumeration to identify the message digest (hash) algorithm that was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
hashNone	No hash algorithm has been selected. This is not a secure connection with the server.	0
hashMD5	The MD5 algorithm was selected. This algorithm produces a 128-bit message digest. This algorithm is no longer considered to be cryptographically secure.	1
hashSHA	The SHA-1 algorithm was selected. This algorithm produces a 160-bit message digest. This algorithm is no longer considered to be cryptographically secure.	2
hashSHA256	The SHA-256 algorithm was selected. This algorithm produces a 256-bit message digest.	4
hashSHA384	The SHA-384 algorithm was selected. This algorithm produces a 384-bit message digest.	8
hashSHA512	The SHA-512 algorithm was selected. This algorithm produces a 512-bit message digest.	16

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

SocketTools Namespace

HttpClient.SecureKeyAlgorithm Enumeration

Specifies the key exchange algorithms that the HttpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum HttpClient.SecureKeyAlgorithm
```

[C#]

```
[Flags]
public enum HttpClient.SecureKeyAlgorithm
```

Remarks

The HttpClient class uses the **SecureKeyAlgorithm** enumeration to identify the key exchange algorithm that was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
keyExchangeNone	No key exchange algorithm has been selected. This is not a secure connection with the server.	0
keyExchangeRSA	The RSA public key exchange algorithm has been selected.	1
keyExchangeKEA	The KEA public key exchange algorithm has been selected. This is an improved version of the Diffie-Hellman public key algorithm.	2
keyExchangeDH	The Diffie-Hellman public key exchange algorithm has been selected.	4
keyExchangeECDH	The Elliptic Curve Diffie-Hellman key exchange algorithm was selected. This is a variant of the Diffie-Hellman algorithm which uses elliptic curve cryptography. This key exchange algorithm is only supported on Windows XP SP3 SP3 and later versions of the operating system.	8

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

SocketTools Namespace

HttpClient.SecurityCertificate Enumeration

Specifies the security certificate status values that may be returned by the HttpClient class.

[Visual Basic]
```
Public Enum HttpClient.SecurityCertificate
```

[C#]
```
public enum HttpClient.SecurityCertificate
```

Remarks

The HttpClient class uses the **SecurityCertificate** enumeration to identify the current status of the certificate that was provided by the remote host when a secure connection was established.

Members

Member Name	Description
certificateNone	No certificate information is available. A secure connection was not established with the server.
certificateValid	The certificate is valid.
certificateNoMatch	The certificate is valid, however the domain name specified in the certificate does not match the domain name of the remote host. The application can examine the **CertificateSubject** property to determine the site the certificate was issued to.
certificateExpired	The certificate has expired and is no longer valid. The application can examine the **CertificateExpires** property to determine when the certificate expired.
certificateRevoked	The certificate has been revoked and is no longer valid. It is recommended that the application immediately terminate the connection if this status is returned.
certificateUntrusted	The certificate has not been issued by a trusted authority, or the certificate is not trusted on the local host. It is recommended that the application immediately terminate the connection if this status is returned.
certificateInvalid	The certificate is invalid. This typically indicates that the internal structure of the certificate is damaged. It is recommended that the application immediately terminate the connection if this status is returned.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

SocketTools Namespace

HttpClient.SecurityProtocols Enumeration

Specifies the security protocols that the HttpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum HttpClient.SecurityProtocols
```

[C#]
```
[Flags]
public enum HttpClient.SecurityProtocols
```

Remarks

The HttpClient class uses the **SecurityProtocols** enumeration to specify one or more security protocols to be used when establishing a connection with a remote host. Multiple protocols may be specified if necessary and the actual protocol used will be negotiated with the remote host. It is recommended that most applications use **protocolDefault** when creating a secure connection.

Members

Member Name	Description	Value
protocolNone	No security protocol will be used, a secure connection will not be established.	0
protocolSSL2	The SSL 2.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections.	1
protocolSSL3	The SSL 3.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections. In most cases, this protocol is only selected if TLS is not supported by the server.	2
protocolTLS10	The TLS 1.0 protocol has been selected. This version of the protocol is commonly used by older servers and is the only version of TLS supported on Windows platforms prior to Windows 7 SP1 and Windows Server 2008 R2.	4
protocolTLS11	The TLS 1.1 protocol has been selected. This version of TLS is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of the	8

	operating system.	
protocolTLS12	The TLS 1.2 protocol has been selected. This is the default version of the protocol and is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of Windows. It is recommended that you use this version of TLS.	16
protocolSSL	Any version of the Secure Sockets Layer (SSL) protocol should be used. The actual protocol version used will be negotiated with the remote host.	3
protocolTLS	Any version of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with with preference for TLS 1.2.	28
protocolTLS1	Version 1.0, 1.1 or 1.2 of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with preference for TLS 1.2.	28
protocolDefault	The default selection of security protocols will be used when establishing a connection. The TLS 1.2, 1.1 and 1.0 protocols will be negotiated with the host, in that order of preference. This option will always request the latest version of the preferred security protocols and is the recommended value.	16
protocolUnknown	An unknown or unsupported security protocol has been specified. This value indicates an error condition.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

SocketTools Namespace

HttpClient.TraceOptions Enumeration

Specifies the logging options that the HttpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum HttpClient.TraceOptions
```

[C#]

```
[Flags]
public enum HttpClient.TraceOptions
```

Remarks

The HttpClient class uses the **TraceOptions** enumeration to specify what kind of debugging information is written to the trace logfile. These options are only meaningful when trace logging is enabled by setting the **Trace** property to **true**.

Members

Member Name	Description	Value
traceDefault	The default trace logging option. This is the same as specifying the **traceInfo** option.	0
traceInfo	All network function calls are written to the trace file. This is the default value.	0
traceError	Only those network function calls which fail are recorded in the trace file.	1
traceWarning	Only those network function calls which fail, or return values which indicate a warning, are recorded in the trace file.	2
traceHexDump	All network function calls are written to the trace file, plus all the data that is sent or received is displayed, in both ASCII and hexadecimal format.	4
traceProcess	All function calls in the current process are logged, rather than only those functions in the current thread. This option is useful for multithreaded applications that are using worker threads.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

SocketTools Namespace

HttpClient.OnCommandEventHandler Delegate

Represents the method that will handle the OnCommand event.

[Visual Basic]

```
Public Delegate Sub HttpClient.OnCommandEventHandler( _
   ByVal sender As Object, _
   ByVal e As CommandEventArgs _
)
```

[C#]

```
public delegate void HttpClient.OnCommandEventHandler(
      object sender,
      CommandEventArgs e
   );
```

Parameters

sender
The source of the event.

e
A CommandEventArgs object that contains the event data.

Remarks

When you create an **OnCommandEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnCommandEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

SocketTools Namespace

HttpClient.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub HttpClient.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void HttpClient.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ErrorEventArgs that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

SocketTools Namespace

HttpClient.OnProgressEventHandler Delegate

Represents the method that will handle the OnProgress event.

[Visual Basic]

```
Public Delegate Sub HttpClient.OnProgressEventHandler( _
   ByVal sender As Object, _
   ByVal e As ProgressEventArgs _
)
```

[C#]

```
public delegate void HttpClient.OnProgressEventHandler(
      object sender,
      ProgressEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A ProgressEventArgs that contains the event data.

Remarks

When you create an **OnProgressEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnProgressEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

SocketTools Namespace

HttpClient.OnRedirectEventHandler Delegate

Represents the method that will handle the OnRedirect event.

[Visual Basic]

```
Public Delegate Sub HttpClient.OnRedirectEventHandler( _
   ByVal sender As Object, _
   ByVal e As RedirectEventArgs _
)
```

[C#]

```
public delegate void HttpClient.OnRedirectEventHandler(
      object sender,
      RedirectEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A RedirectEventArgs that contains the event data.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

SocketTools Namespace

HttpClient.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see HttpClient.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.HttpClient.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class HttpClient.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class HttpClient.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the HttpClient class.

Example

```
<Assembly: SocketTools.HttpClient.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.HttpClient.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

HttpClient.RuntimeLicenseAttribute Members | SocketTools Namespace

HttpClient.RuntimeLicenseAttribute Members

HttpClient.RuntimeLicenseAttribute overview

Public Instance Constructors

HttpClient.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpClient.RuntimeLicenseAttribute Class | SocketTools Namespace

HttpClient.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public HttpClient.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the HttpClient class.

See Also

HttpClient.RuntimeLicenseAttribute Class | SocketTools Namespace

HttpClient.RuntimeLicenseAttribute Properties

The properties of the **HttpClient.RuntimeLicenseAttribute** class are listed below. For a complete list of **HttpClient.RuntimeLicenseAttribute** class members, see the HttpClient.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

HttpClient.RuntimeLicenseAttribute Class | SocketTools Namespace

HttpClient.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

HttpClient.RuntimeLicenseAttribute Class | SocketTools Namespace

HttpClientException Class

The exception that is thrown when a client error occurs.

For a list of all members of this type, see HttpClientException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.HttpClientException

[Visual Basic]
```
Public Class HttpClientException
    Inherits ApplicationException
```

[C#]
```
public class HttpClientException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A HttpClientException is thrown by the HttpClient class when an error occurs.

The default constructor for the HttpClientException class sets the **ErrorCode** property to the last client error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpClient (in SocketTools.HttpClient.dll)

See Also

HttpClientException Members | SocketTools Namespace

HttpClientException Members

HttpClientException overview

Public Instance Constructors

HttpClientException	Overloaded. Initializes a new instance of the HttpClientException class.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpClientException Class | SocketTools Namespace

HttpClientException Constructor

Initializes a new instance of the HttpClientException class with the last network error code.

Overload List

Initializes a new instance of the HttpClientException class with the last network error code.

```
public HttpClientException();
```

Initializes a new instance of the HttpClientException class with a specified error number.

```
public HttpClientException(int);
```

Initializes a new instance of the HttpClientException class with a specified error message.

```
public HttpClientException(string);
```

Initializes a new instance of the HttpClientException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
public HttpClientException(string,Exception);
```

See Also

HttpClientException Class | SocketTools Namespace

HttpClientException Constructor ()

Initializes a new instance of the HttpClientException class with the last network error code.

[Visual Basic]
```
Overloads Public Sub New()
```

[C#]
```
public HttpClientException();
```

Remarks

The ctor constructor sets the **ErrorCode** property to the last client error that occurred. For more information about the errors that may occur, refer to the HttpClient.ErrorCode enumeration.

See Also

HttpClientException Class | SocketTools Namespace | HttpClientException Constructor Overload List

HttpClientException Constructor (String)

Initializes a new instance of the HttpClientException class with a specified error message.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]

```
public HttpClientException(
   string message
);
```

Parameters

message
The error message that explains the reason for the exception.

Remarks

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

HttpClientException Class | SocketTools Namespace | HttpClientException Constructor Overload List

HttpClientException Constructor (String, Exception)

Initializes a new instance of the HttpClientException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal innerException As Exception _
)
```

```
[C#]
public HttpClientException(
   string message,
   Exception innerException
);
```

Parameters

message

The error message that explains the reason for the exception.

innerException

The exception that is the cause of the current exception. If the ***innerException*** parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

HttpClientException Class | SocketTools Namespace | HttpClientException Constructor Overload List

HttpClientException Constructor (Int32)

Initializes a new instance of the HttpClientException class with a specified error number.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

[C#]

```
public HttpClientException(
   int code
);
```

Parameters

code

An integer value which specifies an error code.

Remarks

This constructor sets the **ErrorCode** property to the specified error code. For more information about the errors that may occur, refer to the HttpClient.ErrorCode enumeration.

See Also

HttpClientException Class | SocketTools Namespace | HttpClientException Constructor Overload List

HttpClientException Properties

The properties of the **HttpClientException** class are listed below. For a complete list of **HttpClientException** class members, see the HttpClientException Members topic.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

HttpClientException Class | SocketTools Namespace

HttpClientException.ErrorCode Property

Gets a value which specifies the error that caused the exception.

[Visual Basic]
```
Public ReadOnly Property ErrorCode As ErrorCode
```

[C#]
```
public HttpClient.ErrorCode ErrorCode {get;}
```

Property Value

Returns a HttpClient.ErrorCode enumeration value which specifies the error code.

Remarks

The **ErrorCode** property returns the error code that specifies the cause of the exception.

The default constructor for the HttpClientException class sets the error code to the last network error that occurred. For more information about the errors that may occur, refer to the HttpClient.ErrorCode enumeration.

See Also

HttpClientException Class | SocketTools Namespace

HttpClientException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]

```
Overrides Public ReadOnly Property Message As String
```

[C#]

```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

HttpClientException Class | SocketTools Namespace

HttpClientException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception. This value is the same as the values returned by the Windows Sockets API. For more information about socket error codes, see the Windows Socket Version 2 API error code documentation in MSDN.

See Also

HttpClientException Class | SocketTools Namespace

HttpClientException Methods

The methods of the **HttpClientException** class are listed below. For a complete list of **HttpClientException** class members, see the HttpClientException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpClientException Class | SocketTools Namespace

HttpClientException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]
```
Overrides Public Function ToString() As String
```

[C#]
```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

HttpClientException Class | SocketTools Namespace

HttpServer Class

Implements a server that enables the application to send and receive files using the Hypertext Transfer Protocol.

For a list of all members of this type, see HttpServer Members.

System.Object
  SocketTools.HttpServer

[Visual Basic]
```
Public Class HttpServer
    Implements IDisposable
```

[C#]
```
public class HttpServer : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The **SocketTools.HttpServer** class provides an interface for implementing an embedded, lightweight server that can be used to provide access to documents and other resources using the Hypertext Transfer Protocol. The server can accept connections from any standard web browser, third-party applications or programs developed using the **SocketTools.HttpClient** class.

The application specifies an initial server configuration and then responds to events that are generated when the client sends a request to the server. An application may implement only minimal handlers for most events, in which case the default actions are performed for most standard HTTP commands. However, an application may also use the event mechanism to filter specific commands or to extend the protocol by providing custom implementations of existing commands or add entirely new commands.

The server includes support for CGI scripting, virtual hosting, client authentication and the creation of virtual directories and files. The server also supports secure connections using TLS 1.2. Secure connections require that a valid server certificate be installed on the system.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

HttpServer Members | SocketTools Namespace

HttpServer Members

HttpServer overview

Public Static (Shared) Fields

defaultHost	Defines the ID used to specify the default virtual host.
invalidHost	Defines an invalid virtual host identifier.

Public Static (Shared) Methods

ErrorText	Returns the description of an error code.

Public Instance Constructors

HttpServer Constructor	Initializes a new instance of the HttpServer class.

Public Instance Fields

AdapterAddress	Returns the IP address associated with the specified network adapter.

Public Instance Properties

AdapterCount	Get the number of available local and remote network adapters.
CacheTime	Gets and sets a value that specifies the current cache time period.
CertificateName	Gets and sets a value that specifies the name of the server certificate.
CertificatePassword	Gets and sets the password associated with the server certificate.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateUser	Gets and sets the user that owns the server certificate.
ClientAccess	Gets and sets the access rights that have been granted to the client session.
ClientAddress	Return the Internet address of the current client connection.
ClientCount	Return the number of active client sessions connected to the server.
ClientDirectory	Return the current working directory for the active client session.
ClientHost	Return the host name that the client used to establish the connection.
ClientId	Gets the unique client identifier for the current

	client session.
ClientIdle	Gets and sets the maximum number of seconds a client can be idle before the server terminates the session.
ClientPort	Gets a value that specifies the port number used by the current client session.
ClientThread	Gets the thread ID for the current client session.
ClientUser	Return the user name associated with the specified client session.
CommandLine	Return the complete command line issued by the client.
Directory	Get and set the full path to the root directory assigned to the server.
ExecTime	Get and set maximum number of seconds that the server will permit an external script handler to execute.
ExternalAddress	Return the external IP address for the local system.
HiddenFiles	Determine if the server should permit access to hidden files.
Identity	Gets and sets a string that identifies the server to the client.
IdleTime	Gets and sets the maximum number of seconds a client can be idle before the server terminates the session.
IsActive	Gets a value which indicates if the server is active.
IsAuthenticated	Determine if the active client session has been authenticated.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsListening	Gets a value which indicates if the server is listening for client connections.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalPath	Return the full path to the local file or directory that is the target of the current command.
LocalUser	Determines if the server should perform user authentication using the Windows local account database.
LockFiles	Determines if files should be exclusively locked when a client attempts to upload or download a file.

LogFile	Gets and sets the name of the server log file.
LogFormat	Gets and sets the format used when updating the server log file.
LogLevel	Gets and sets the level of detail included in the server log file.
MaxClients	Gets and sets the maximum number of clients that can connect to the server.
MemoryUsage	Gets the amount of unmanaged memory currently allocated by the server.
MultiUser	Determine if the server should be started in multi-user mode.
NoIndex	Determine if the server should search for a default index page.
Options	Gets and sets the options that may be specified for the server instance.
Priority	Gets and sets a value which specifies the server priority.
ReadOnly	Determine if the server should prevent clients from uploading files.
Restricted	Determine if the server should be started in restricted mode, limiting client access to the server.
Secure	Determine if the server should accept secure client connections.
ServerAddress	Gets and sets the address that will be used by the server to listen for connections.
ServerHandle	Gets the handle to the server created to listen for client connections.
ServerName	Gets a value which specifies the host name for the local system.
ServerPort	Gets and sets the port number that will be used by the server to listen for connections.
ServerThread	Gets the thread ID for the current server.
ServerUuid	Gets and sets the Universally Unique Identifier (UUID) associated with the server.
StackSize	Gets and sets the size of the stack allocated for threads created by the server.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of

	the network function tracing logfile.
TraceFlags	Gets and sets a value which specifies the network function tracing flags.
Version	Gets a value which returns the current version of the HttpServer class library.
VirtualPath	Return the virtual path to the local file or directory that is the target of the current command.

Public Instance Methods

AddHost	Overloaded. Add a new virtual host to the server virtual host table.
AddPath	Overloaded. Add a new virtual path for the specified host.
AddUser	Overloaded. Add a new virtual user to the specified host.
Authenticate	Overloaded. Authenticate the client and assign access rights for the session.
CheckPath	Overloaded. Determine if the client has permission to access the specified virtual path.
ClearHeaders	Overloaded. Delete all of the response headers for the specified client session.
DeleteHost	Overloaded. Delete a virtual host associated with the specified server.
DeletePath	Overloaded. Delete a virtual path from the specified virtual host.
DeleteUser	Overloaded. Remove a virtual user from the server.
Disconnect	Overloaded. Disconnect the specified client session from the server.
Dispose	Overloaded. Releases all resources used by HttpServer.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetAllHeaders	Overloaded. Return all of the request header values in the specified string buffer.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetHeader	Overloaded. Return the value of a request header for the specified client session.
GetType (inherited from Object)	Gets the Type of the current instance.
GetVariable	Overloaded. Return the value of a CGI environment variable for the specified client.

Initialize	Overloaded. Initialize an instance of the HttpServer class.
ReceiveRequest	Overloaded. Receive the request that was sent by the client to the server.
RedirectRequest	Overloaded. Redirect the request from the client to another URL.
RegisterHandler	Overloaded. Register a CGI program for use and associate it with a file name extension.
RegisterProgram	Overloaded. Register a CGI program for use and associate it with a virtual path on the server.
RequireAuthentication	Overloaded. Send a response to the client indicating that authentication is required.
Reset	Reset the internal state of the object, resetting all properties to their default values.
ResolvePath	Overloaded. Resolve a path to its full virtual or local file name.
Restart	Restarts the server and terminates all active client connections.
Resume	Resume accepting new client connections.
SendError	Overloaded. Send an error result code and message to the client in response to a command.
SendResponse	Overloaded. Send a result code and message to the client in response to a command.
SetHeader	Overloaded. Create or change the value of a response header for the client session.
SetVariable	Overloaded. Create or change the value of a CGI environment variable for the specified client.
Start	Overloaded. Start listening for client connections on the specified IP address and port number.
Stop	Stop listening for new client connections and terminate all active clients already connected to the server.
Suspend	Suspend accepting new client connections.
Throttle	Overloaded. Limit the maximum number of client connections, connections per IP address and connection rate.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the server.

Public Instance Events

OnAuthenticate	Occurs when the client has requested

	authentication with the specified username and password.
OnCommand	Occurs when a client has issued a command to the server.
OnConnect	Occurs when a connection is established with the remote host.
OnDisconnect	Occurs when the remote host disconnects from the local system.
OnDownload	Occurs when a connection is established with the remote host.
OnError	Occurs when an network operation fails.
OnExecute	Occurs when the client has executed an external script handler on the server.
OnIdle	Occurs when the there are no clients connected to the server.
OnResult	Occurs when the command issued by the client has been processed by the server.
OnStart	Occurs when the server starts accepting connections.
OnStop	Occurs when the server stops accepting connections.
OnTimeout	Occurs when the client has exceeded the maximum allowed idle time.
OnUpload	Occurs when the client has successfully uploaded a file to the server.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the HttpServer class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpServer Class | SocketTools Namespace

HttpServer Constructor

Initializes a new instance of the HttpServer class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public HttpServer();
```

See Also

HttpServer Class | SocketTools Namespace

HttpServer Fields

The fields of the **HttpServer** class are listed below. For a complete list of **HttpServer** class members, see the HttpServer Members topic.

Public Static (Shared) Fields

defaultHost	Defines the ID used to specify the default virtual host.
invalidHost	Defines an invalid virtual host identifier.

Public Instance Fields

AdapterAddress	Returns the IP address associated with the specified network adapter.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.AdapterAddress Field

Returns the IP address associated with the specified network adapter.

[Visual Basic]

```
Public ReadOnly AdapterAddress As AdapterAddressArray
```

[C#]

```
public readonly AdapterAddressArray AdapterAddress;
```

Remarks

The **AdapterAddress** array returns the IP addresses that are associated with the local network or remote dial-up network adapters configured on the system. The **AdapterCount** property can be used to determine the number of adapters that are available.

Multihomed systems with more than one local network adapter, or a combination of local and dial-up adapters will not be listed in a specific order. An application should not make the assumption that the first address returned by **AdapterAddress** always refers to a local network adapter.

Note that it is possible that the **AdapterCount** property will return 0, and **AdapterAddress** will return an empty string. This indicates that the system does not have a physical network adapter with an assigned IP address, and there are no dial-up networking connections currently active. If a dial-up networking connection is established at some later point, the **AdapterCount** property will change to 1, and the **AdapterAddress** property will return the IP address allocated for that connection.

See Also

HttpServer Class | SocketTools Namespace | AdapterAddressArray Class | AdapterCount Property

HttpServer Properties

The properties of the **HttpServer** class are listed below. For a complete list of **HttpServer** class members, see the HttpServer Members topic.

Public Instance Properties

AdapterCount	Get the number of available local and remote network adapters.
CacheTime	Gets and sets a value that specifies the current cache time period.
CertificateName	Gets and sets a value that specifies the name of the server certificate.
CertificatePassword	Gets and sets the password associated with the server certificate.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateUser	Gets and sets the user that owns the server certificate.
ClientAccess	Gets and sets the access rights that have been granted to the client session.
ClientAddress	Return the Internet address of the current client connection.
ClientCount	Return the number of active client sessions connected to the server.
ClientDirectory	Return the current working directory for the active client session.
ClientHost	Return the host name that the client used to establish the connection.
ClientId	Gets the unique client identifier for the current client session.
ClientIdle	Gets and sets the maximum number of seconds a client can be idle before the server terminates the session.
ClientPort	Gets a value that specifies the port number used by the current client session.
ClientThread	Gets the thread ID for the current client session.
ClientUser	Return the user name associated with the specified client session.
CommandLine	Return the complete command line issued by the client.
Directory	Get and set the full path to the root directory assigned to the server.

ExecTime	Get and set maximum number of seconds that the server will permit an external script handler to execute.
ExternalAddress	Return the external IP address for the local system.
HiddenFiles	Determine if the server should permit access to hidden files.
Identity	Gets and sets a string that identifies the server to the client.
IdleTime	Gets and sets the maximum number of seconds a client can be idle before the server terminates the session.
IsActive	Gets a value which indicates if the server is active.
IsAuthenticated	Determine if the active client session has been authenticated.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsListening	Gets a value which indicates if the server is listening for client connections.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalPath	Return the full path to the local file or directory that is the target of the current command.
LocalUser	Determines if the server should perform user authentication using the Windows local account database.
LockFiles	Determines if files should be exclusively locked when a client attempts to upload or download a file.
LogFile	Gets and sets the name of the server log file.
LogFormat	Gets and sets the format used when updating the server log file.
LogLevel	Gets and sets the level of detail included in the server log file.
MaxClients	Gets and sets the maximum number of clients that can connect to the server.
MemoryUsage	Gets the amount of unmanaged memory currently allocated by the server.
MultiUser	Determine if the server should be started in multi-user mode.
NoIndex	Determine if the server should search for a default index page.

Options	Gets and sets the options that may be specified for the server instance.
Priority	Gets and sets a value which specifies the server priority.
ReadOnly	Determine if the server should prevent clients from uploading files.
Restricted	Determine if the server should be started in restricted mode, limiting client access to the server.
Secure	Determine if the server should accept secure client connections.
ServerAddress	Gets and sets the address that will be used by the server to listen for connections.
ServerHandle	Gets the handle to the server created to listen for client connections.
ServerName	Gets a value which specifies the host name for the local system.
ServerPort	Gets and sets the port number that will be used by the server to listen for connections.
ServerThread	Gets the thread ID for the current server.
ServerUuid	Gets and sets the Universally Unique Identifier (UUID) associated with the server.
StackSize	Gets and sets the size of the stack allocated for threads created by the server.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the network function tracing logfile.
TraceFlags	Gets and sets a value which specifies the network function tracing flags.
Version	Gets a value which returns the current version of the HttpServer class library.
VirtualPath	Return the virtual path to the local file or directory that is the target of the current command.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.AdapterCount Property

Get the number of available local and remote network adapters.

[Visual Basic]
```
Public ReadOnly Property AdapterCount As Integer
```

[C#]
```
public int AdapterCount {get;}
```

Property Value

Returns the number of available local and remote network adapters.

Remarks

The **AdapterCount** property returns the number of local and remote dial-up networking adapters available on the local system. This value can be used in conjunction with the **AdapterAddress** array to enumerate the IP addresses assigned to the various network adapters.

Note that it is possible that the **AdapterCount** property will return 0, and **AdapterAddress** will return an empty string. This indicates that the system does not have a physical network adapter with an assigned IP address, and there are no dial-up networking connections currently active. If a dial-up networking connection is established at some later point, the **AdapterCount** property will change to 1, and the **AdapterAddress** property will return the IP address allocated for that connection.

See Also

HttpServer Class | SocketTools Namespace | AdapterAddress Field

HttpServer.CertificateName Property

Gets and sets a value that specifies the name of the server certificate.

[Visual Basic]

```
Public Property CertificateName As String
```

[C#]

```
public string CertificateName {get; set;}
```

Property Value

A string which specifies the server certificate name.

Remarks

The **CertificateName** property sets the common name or friendly name of the certificate that should be used when starting a secure server. If the **Secure** property is set to True, this property must be specify a valid certificate name. The certificate must have a private key associated with it, otherwise client connections will fail because the class will be unable to create a security context for the session.

When the certificate store is searched for a matching certificate, it will first search for any certificate with a friendly name that matches the property value. If no valid certificate is found, it will then search for a certificate with a matching common name.

Certificates may be installed and viewed on the local system using the Certificate Manager that is included with the Windows operating system. For more information, refer to the documentation for the Microsoft Management Console.

See Also

HttpServer Class | SocketTools Namespace | CertificateStore Property | Secure Property

HttpServer.CertificateStore Property

Gets and sets a value that specifies the name of the local certificate store.

[Visual Basic]

```
Public Property CertificateStore As String
```

[C#]

```
public string CertificateStore {get; set;}
```

Property Value

A string which specifies the certificate store name. The default value is the current user's personal certificate store.

Remarks

The **CertificateStore** property is used to specify the name of the certificate store which contains the security certificate to use when security is enabled for the server. The certificate may either be stored in the registry or in a file. If the certificate is stored in the registry, then this property should be set to one of the following predefined values:

Store Name	Description
CA	Certification authority certificates. These are certificates that are issued by entities which are entrusted to issue certificates to other individuals or organizations. Companies such as VeriSign and Thawte act as certification authorities.
MY	Personal certificates and their associated private keys for the current user. This store typically holds the client certificates used to establish a user's credentials. If a certificate store is not specified, this is the default value that is used.
ROOT	Certificates that have been self-signed by a certificate authority. Root certificates for a number of different certification authorities such as VeriSign and Thawte are installed as part of the operating system and periodically updated by Microsoft.

In most cases the server certificate will be installed in the user's personal certificate store, and therefore it is not necessary to set this property value because that is the default location that will be used to search for the certificate. This property is only used if the **CertificateName** property is also set to a valid certificate name.

If you are using a local certificate store, with the certificate and private key stored in the registry, you can explicitly specify whether the certificate store for the current user or the local machine (all users) should be used. This is done by prefixing the certificate store name with "HKCU" for the current user, or "HKLM" for the local machine. For example, a certificate store name of "HKLM:MY" would specify the personal certificate store for the local machine, rather than the current user. If neither prefix is specified, it will default to the certificate store for the current user.

This property may also be used to specify a file that contains the server certificate. In this case, the property should specify the full path to the file and must contain both the certificate and private key in

PKCS12 format. If the file is protected by a password, the **CertificatePassword** property must also be set to specify the password.

See Also

HttpServer Class | SocketTools Namespace | CertificatePassword Property | Secure Property

HttpServer.ClientAccess Property

Gets and sets the access rights that have been granted to the client session.

[Visual Basic]

```
Public Property ClientAccess As UserAccess
```

[C#]

```
public HttpServer.UserAccess ClientAccess {get; set;}
```

Property Value

A UserAccess enumeration that specifies on or more user access permissions.

Remarks

The **ClientAccess** property is used to determine all of the access permissions that are currently granted to an authenticated client session and optionally change those permissions. For a list of user access rights that can be granted to the client, see the UserAccess enumeration.

When modifying the value of this property, it is recommended that you use bitwise OR and AND operands to set and clear specific bitflags. The exception is when using the **httpAccessDefault** permission. If you wish to reset the client session to use the default permissions based on the server configuration and client authentication, then you should assign this value directly to the **ClientAccess** property.

This property should only be accessed within an event handler such as **OnCommand** because its value is specific to the client session that raised the event. This property will always return a value of zero outside of an event handler, and an exception will be raised if you attempt to modify this property outside of an event handler.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.ClientAddress Property

Return the Internet address of the current client connection.

[Visual Basic]

```
Public ReadOnly Property ClientAddress As String
```

[C#]

```
public string ClientAddress {get;}
```

Property Value

A string that specifies the client Internet Protocol address.

Remarks

The **ClientAddress** property returns the address of the current client session which has connected to the server. This property should only be accessed within an event handler such as **OnConnect** because its value is specific to the client session that raised the event. This property will always return an empty string when accessed outside of an event handler.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.ClientCount Property

Return the number of active client sessions connected to the server.

[Visual Basic]
```
Public ReadOnly Property ClientCount As Integer
```

[C#]
```
public int ClientCount {get;}
```

Property Value

An integer value that specifies the number of active client sessions.

Remarks

The **ClientCount** read-only property returns the number of active client sessions that have been established with the server. The value includes both authenticated and unauthenticated client sessions.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.ClientDirectory Property

Return the current working directory for the active client session.

[Visual Basic]
```
Public ReadOnly Property ClientDirectory As String
```

[C#]
```
public string ClientDirectory {get;}
```

Property Value

A string that specifies the full path to a local directory on the server.

Remarks

The **ClientDirectory** property returns the current working directory for the active client session. Initially this value will be the absolute path on the local system that maps to an authenticated client's home directory. The client can change its current working directory using the CWD command. The **ClientHome** property will return the home directory that has been assigned to the client.

It is important to note that the current working directory for client sessions is virtual, and does not reflect the current working directory for the server process. This property should only be accessed within an event handler after the client session has been authenticated. Unauthenticated clients are not assigned a current working directory. This property will always return an empty string when accessed outside of an event handler.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.ClientHost Property

Return the host name that the client used to establish the connection.

[Visual Basic]
```
Public ReadOnly Property ClientHost As String
```

[C#]
```
public string ClientHost {get;}
```

Property Value

A string that specifies the host name used by the client to connect to the server.

Remarks

The **ClientHost** property returns the host name that the client used to establish the connection. If the client does not explicitly specify the host name, then this property will return the same host name that was assigned to the server when it started.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.ClientId Property

Gets the unique client identifier for the current client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which uniquely identifies the client session.

Remarks

Each client connection that is accepted by the server is assigned a unique numeric value. This value can be used by the application to identify that client session, and is different than the socket handle allocated for the client. While it is possible for a client socket handle to be reused by the operating system, client IDs are unique throughout the life of the server session and are never duplicated.

It is important to note that the actual value of the client ID should be considered opaque. It is only guaranteed that the value will be greater than zero, and that it will be unique to the client session.

This property only returns a meaningful value when accessed from within a class event handler, or a method that has been invoked from within an event handler.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.ClientIdle Property

Gets and sets the maximum number of seconds a client can be idle before the server terminates the session.

[Visual Basic]

```
Public Property ClientIdle As Integer
```

[C#]

```
public int ClientIdle {get; set;}
```

Property Value

An integer value that specifies the idle timeout period in seconds.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.ClientPort Property

Gets a value that specifies the port number used by the current client session.

[Visual Basic]

```
Public ReadOnly Property ClientPort As Integer
```

[C#]

```
public int ClientPort {get;}
```

Property Value

An integer value which specifies the peer port number.

Remarks

The **ClientPort** property returns the port number that the current client has used when establishing a connection with the server. This property value is only meaningful when accessed within an event handler such as the **OnConnect** event.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.ClientThread Property

Gets the thread ID for the current client session.

[Visual Basic]

```
Public ReadOnly Property ClientThread As Integer
```

[C#]

```
public int ClientThread {get;}
```

Property Value

An integer value which identifies the client thread that was created to manage the client session.

Remarks

Until the thread terminates, the thread identifier uniquely identifies the thread throughout the system.

This property only returns a meaningful value when accessed from within a class event handler, or a method that has been invoked from within an event handler.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.ClientUser Property

Return the user name associated with the specified client session.

[Visual Basic]

```
Public ReadOnly Property ClientUser As String
```

[C#]

```
public string ClientUser {get;}
```

Property Value

A string that specifies the user name associated with the active client session.

Remarks

The **ClientUser** property returns the user name that the client used to authenticate the client session. This property should only be accessed within an event handler after the client session has been authenticated. Unauthenticated clients are not assigned a user name. This property will always return an empty string when accessed outside of an event handler.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.Directory Property

Get and set the full path to the root directory assigned to the server.

[Visual Basic]

```
Public Property Directory As String
```

[C#]

```
public string Directory {get; set;}
```

Property Value

A string that specifies the full path to a local directory on the server.

Remarks

The **Directory** property returns the path to the root directory for the server. If this property is set to the name of a valid directory before the server is started, that directory will be considered the root directory for the server. If this property is not set, or is set to an empty string, then the server will use the current working directory as its root directory, however this is not recommended. It is recommended that you specify an absolute path to the directory, otherwise the path will be relative to the current working directory. You may include environment variables in the path surrounded by percent (%) symbols and they will be expanded.

If you have configured the server to permit clients to upload files, you must ensure that your application has permission to create files in the directory that you specify. A recommended location for the server root directory would be a subdirectory of the %ALLUSERSPROFILE% directory. Using the environment variable ensures that your server will work correctly on different versions of Windows. If the root directory does not exist at the time that the server is started, it will be created.

If the **MultiUser** property is **False**, all authenticated clients will have their current working directory initialized to the server root directory. If the **MultiUser** property is **True**, then the Public and User subdirectories will be created in the root directory, and each authenticated client will have their current working directory initialized to their individual home directory.

This property can be read after the server has started and it will return the full path to the root directory. However, attempting to change the value of this property after the server has started will cause an exception to be raised. To change the root directory for the server, you must first call the **Stop** method which will terminate all active client connections.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.ExternalAddress Property

Return the external IP address for the local system.

[Visual Basic]

```
Public ReadOnly Property ExternalAddress As String
```

[C#]

```
public string ExternalAddress {get;}
```

Property Value

A string that specifies the external Internet Protocol address for the local system.

Remarks

The **ExternalAddress** property returns the IP address assigned to the router that connects the local host to the Internet. This is typically used by an application executing on a system in a local network that uses a router which performs Network Address Translation (NAT). In that network configuration, the **ServerAddress** property will only return the IP address for the local system on the LAN side of the network. The **ExternalAddress** property can be used to determine the IP address assigned to the router on the Internet side of the connection and can be particularly useful for servers running on a system behind a NAT router. Note that you should not assign the **ServerAddress** property to the value returned by the **ExternalAddress** property. If the server is running behind a NAT router, the router must be configured to forward incoming connections to the appropriate address on the LAN.

Using this property requires that you have an active connection to the Internet; checking the value of this property on a system that uses dial-up networking may cause the operating system to automatically connect to the Internet service provider. The control may be unable to determine the external IP address for the local host for a number of reasons, particularly if the system is behind a firewall or uses a proxy server that restricts access to external sites on the Internet. If the external address for the local host cannot be determined, the property will return an empty string.

If the control is able to obtain a valid external address for the local host, that address will be cached for sixty minutes. Because dial-up connections typically have different IP addresses assigned to them each time the system is connected to the Internet, it is recommended that this property only be used in conjunction with persistent broadband connections.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.HiddenFiles Property

Determine if the server should permit access to hidden files.

[Visual Basic]
```
Public Property HiddenFiles As Boolean
```

[C#]
```
public bool HiddenFiles {get; set;}
```

Property Value

A Boolean value that specifies if hidden files can be accessed by clients.

Remarks

The **HiddenFiles** property determines if the server should allow clients to access files with the hidden and/or system attribute. If this property is **True**, then hidden files are included in directory listings and clients may download or replace hidden files. If the property is **False**, hidden files are not included in directory listings and any attempt to access, delete or modify a hidden file will result in an error.

The default value for this property is **False.**

See Also

HttpServer Class | SocketTools Namespace

HttpServer.Identity Property

Gets and sets a string that identifies the server to the client.

[Visual Basic]

```
Public Property Identity As String
```

[C#]

```
public string Identity {get; set;}
```

Property Value

A string that identifies the server instance.

Remarks

The **Identity** property returns a string that is used to identify the server. It is used for informational purposes only and does not affect the operation of the server. Typically the string specifies the name of the application and a version number, and is displayed whenever a client establishes its initial connection to the server. This property can be set to assign an identity to the server, however after the server has started this property becomes read-only.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.IdleTime Property

Gets and sets the maximum number of seconds a client can be idle before the server terminates the session.

[Visual Basic]

```
Public Property IdleTime As Integer
```

[C#]

```
public int IdleTime {get; set;}
```

Property Value

An integer value that specifies the idle timeout period in seconds.

Remarks

The **IdleTime** property specifies the maximum number of seconds that a client session may be idle before the server closes the control connection to the client. A value of zero specifies the default value of 60 seconds. If the value is non-zero, the minimum value is 10 seconds and the maximum value is 300 seconds (5 minutes). This value is used to initialize the default idle timeout period for each client session. The server determines if a client is idle based on the time the last command was issued and whether or not a data transfer is in progress.

The **ClientIdle** property can be used to determine the idle timeout period for a specific client. When the timeout period for the client has elapsed, the **OnTimeout** event will fire prior to the client being disconnected from the server.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.IsActive Property

Gets a value which indicates if the server is active.

[Visual Basic]

```
Public ReadOnly Property IsActive As Boolean
```

[C#]

```
public bool IsActive {get;}
```

Property Value

A Boolean value that specifies if the server instance is currently active.

Remarks

The **IsActive** property returns **True** if the server has been started using the **Start** method. If the server has not been started, the property will return **False**.

To determine if the server is accepting client connections, use the **IsListening** property. This property will only indicate if the server has been started. For example, if the server has been suspended using the **Suspend** method, this property will return a value of **True**, while the **IsListening** property will return a value of **False**.

An application should not depend on this property returning **False** immediately after the **Stop** method has been called to shutdown the server. This property will continue to return **True** until all clients have disconnected from the server and the server thread has terminated. To determine when the server has stopped, implement a handler for the **OnStop** event.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.IsAuthenticated Property

Determine if the active client session has been authenticated.

[Visual Basic]

```
Public ReadOnly Property IsAuthenticated As Boolean
```

[C#]

```
public bool IsAuthenticated {get;}
```

Property Value

A Boolean value that specifies if the active client session has been authenticated.

Remarks

The **IsAuthenticated** property returns **True** if the active client session has successfully authenticated with a valid username and password. This property should only be accessed within an event handler such as **OnCommand** because its value is specific to the client session that raised the event. This property will always return a value of **False** outside of an event handler.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.IsInitialized Property

Gets a value which indicates if the current instance of the class has been initialized successfully.

[Visual Basic]

```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]

```
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the class instance has been initialized; otherwise returns **false**.

Remarks

The **IsInitialized** property is used to determine if the current instance of the class has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the class may not be able to initialize itself.

The most common reasons that a class instance may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required networking libraries or not being able to access the system registry.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.IsListening Property

Gets a value which indicates if the server is listening for client connections.

[Visual Basic]

```
Public ReadOnly Property IsListening As Boolean
```

[C#]

```
public bool IsListening {get;}
```

Property Value

Returns **true** if the server is listening for client connections; otherwise returns **false**.

Remarks

The **IsListening** property will return **true** if the **Start** method was called and the server is currently accepting incoming client connections. In all other situations, this property will return **false**.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public HttpServer.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.LocalPath Property

Return the full path to the local file or directory that is the target of the current command.

[Visual Basic]
```
Public Property LocalPath As String
```

[C#]
```
public string LocalPath {get; set;}
```

Property Value

A string that specifies the full path to a local file or directory on the server.

Remarks

The **LocalPath** property returns the full path to a local file name or directory specified by the client as an argument to a standard HTTP command. For example, if the client sends the GET command to the server, this property will return the complete path to the local file that the client wants to download. This property will only return a value for those standard commands that perform some action on a file or directory, otherwise it will return an empty string.

Setting this property allows you to effectively redirect the client to use a different file than the one that was actually requested. If the path is absolute, then it will be used as-is. If the path is relative, it will be relative to the current working directory for the active client session. The full path to this file is not limited to the server root directory or its subdirectory, it can specify a file anywhere on the local system. If this property is set to an empty string, then the server will revert to using the actual file or directory name specified by the command.

This property should only be set within an **OnCommand** event handler, and only for those commands that perform an action on a file or directory. If the current command does not target a file or directory, setting this property will cause an exception to be raised by the control. Exercise caution when using this property to redirect the server to use a different file than the one requested by the client; changing the target file may cause the client to behave in unexpected ways.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.LocalUser Property

Determines if the server should perform user authentication using the Windows local account database.

[Visual Basic]
```
Public Property LocalUser As Boolean
```

[C#]
```
public bool LocalUser {get; set;}
```

Property Value

A Boolean value that specifies if the server should authenticate local users.

Remarks

The **LocalUser** property determines if the server should perform user authentication using the Windows local account database. If this option is not specified, the application is responsible for creating virtual users using the **AddUser** method or implementing an **OnAuthenticate** event handler and authenticating client sessions individually.

If this property is set to **True**, a client can authenticate as a local user, however the session will not inherit that user's access rights. All files that are accessed or created by the server will continue to use the permissions of the process that started the server. For example, consider a server application that was started by local user ***A***. Next, a client connects to the server and authenticates itself as local user ***B***. When that client uploads a file to the server, the file that is created will be owned by user ***A***, not user ***B***. This ensures that the server application retains ownership and control of the files that have been created or modified.

The default value for this property is **False**.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.LockFiles Property

Determines if files should be exclusively locked when a client attempts to upload or download a file.

[Visual Basic]
```
Public Property LockFiles As Boolean
```

[C#]
```
public bool LockFiles {get; set;}
```

Property Value

A Boolean value that specifies if files should be locked during file transfers.

Remarks

The **LocalTime** property determines if files should be exclusively locked when a client attempts to upload or download a file. If another client attempts to access the same file, the operation will fail. By default, the server will permit multiple clients to access the same file, although it will still write-lock files that are in the process of being uploaded..

The default value for this property is **False**.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.LogFile Property

Gets and sets the name of the server log file.

[Visual Basic]
```
Public Property LogFile As String
```

[C#]
```
public string LogFile {get; set;}
```

Property Value

A string that specifies the full path to a local log file.

Remarks

The **LogFile** property is used to specify the name of a file that will contain a log of all client activity. The **LogFormat** and **LogLevel** properties affect the specific format for the file and the level of detail included in the log. It is recommended that you specify an absolute path to the log file, otherwise the path will be relative to the current working directory. You may include environment variables in the path surrounded by percent (%) symbols and they will be expanded.

If the log file does not exist it will be created when the server is started. If file already exists, the server will append the new logging data to the file. The server must have permission to create and/or modify the specified file.

Setting this property to an empty string after the server has been started will have the effect of disabling logging, setting the logging level to 0 and the logging format to **FormatType.formatNone**.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.LogFormat Property

Gets and sets the format used when updating the server log file.

[Visual Basic]

```
Public Property LogFormat As FormatType
```

[C#]

```
public HttpServer.FormatType LogFormat {get; set;}
```

Property Value

A FormatType enumeration that specifies the log file format.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.LogLevel Property

Gets and sets the level of detail included in the server log file.

[Visual Basic]
```
Public Property LogLevel As Integer
```

[C#]
```
public int LogLevel {get; set;}
```

Property Value

An integer value that specifies the amount of information the server writes to the log file.

Remarks

The **LogLevel** property is used to specify the level of detail that should generated in the log file. The minimum value is 1 and the maximum value is 10. If this parameter is zero, it is the same as specifying a log file format of **httpLogNone** and will disable logging by the server.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.MaxClients Property

Gets and sets the maximum number of clients that can connect to the server.

[Visual Basic]

```
Public Property MaxClients As Integer
```

[C#]

```
public int MaxClients {get; set;}
```

Property Value

An integer value which specifies the maximum number of client sessions that will be accepted by the server. A value of zero specifies that there is no fixed limit to the maximum number of clients.

Remarks

The **MaxClients** property specifies the maximum number of client connections that will be accepted by the server. Once the maximum number of connections has been established, the server will reject any subsequent connections until the number of active client connections drops below the specified value. A value of zero specifies that there should be no limit on the number of clients.

Changing the value of this property while a server is actively listening for connections will modify the maximum number of client connections permitted, but it will not affect connections that have already been established.

By default, there are no limits on the number of client connections or the connection rate when a server is started. Use the **Throttle** method to change the maximum number of client connections per IP address or the overall connection rate threshold for the server.

It is important to note that regardless of the maximum number of clients specified by this property, the actual number of client connections that can be managed by the server depends on the number of sockets that can be allocated from the operating system. The amount of physical memory installed on the system affects the number of connections that can be maintained because each connection allocates memory for the socket context from the non-paged memory pool.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.MemoryUsage Property

Gets the amount of unmanaged memory currently allocated by the server.

[Visual Basic]

```
Public ReadOnly Property MemoryUsage As Long
```

[C#]

```
public long MemoryUsage {get;}
```

Property Value

A long integer which specifies the number of bytes of memory allocated.

Remarks

This read-only property returns the amount of memory allocated by the server and all active client sessions. It enumerates all unmanaged memory allocations made by the server process and client session threads, returning the total number of bytes allocated for the server. This value reflects the amount of memory explicitly allocated by the class and does not reflect the total working set size of the process, or the memory allocated on the managed heap which is used by the .NET garbage collector.

Getting the value of this property forces the server into a locked state, and all client sessions will block while the memory usage is being calculated. Because this enumerates all unmanaged heaps allocated for the server process, it can be an expensive operation, particularly when there are a large number of active clients connected to the server. Frequently checking the value of this property can significantly degrade the performance of the server. It is primarily intended for use as a debugging tool to determine if memory usage is the result of an increase in active client sessions. If the value returned by this property remains reasonably constant, but the amount of memory allocated for the process continues to grow, it could indicate a memory leak in some other area of the application.

See Also

HttpServer Class | SocketTools Namespace | StackSize Property

HttpServer.MultiUser Property

Determine if the server should be started in multi-user mode.

[Visual Basic]

```
Public Property MultiUser As Boolean
```

[C#]

```
public bool MultiUser {get; set;}
```

Property Value

A Boolean value that specifies if the server should be started in multi-user mode.

Remarks

The **MultiUser** property determines if the server should be started in multi-user mode. If this property is set to **True**, each user will be assigned their own home directory which will be based on their user name. When a client authenticates as that user, its current working directory is set to the user's home directory. If this property is set to **False**, then all users will share the server root directory by default. This property does not affect the maximum number of simultaneous client connections to the server. To isolate users to their own individual home directory, set the **Restricted** property to **True**.

Setting this property to **True** will cause the server to create two subdirectories under the server root directory named Public and Users. The Public subdirectory is where public files should be stored, and also serves as the home directory for anonymous (guest) users. The Users subdirectory is where the home directories for each user will be created.

Attempting to change the value of this property after the server has started will cause an exception to be raised. To change this property value, you must first call the **Stop** method which will terminate all active client connections.

The default value for this property is **False**.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.NoIndex Property

Determine if the server should search for a default index page.

[Visual Basic]

```
Public Property NoIndex As Boolean
```

[C#]

```
public bool NoIndex {get; set;}
```

Property Value

A boolean value that specifies if the server should search for default index pages. The default value for this property is **false**.

Remarks

The **NoIndex** property determines if the server should should search for a default index file if the client requests a resource that maps to a local directory on the server. If this property is set to **true**, the server will not search for an index file. If this property is set to **false**, the server will search for a file named index.htm, index.html, default.htm, default.html or index.txt in the directory. If a file by one of those names is found, it will return the contents of that file rather than a list of files in the directory.

Attempting to change the value of this property after the server has started will cause an exception to be raised. To change this property value, you must first call the **Stop** method which will terminate all active client connections.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.Options Property

Gets and sets the options that may be specified for the server instance.

[Visual Basic]
```
Public Property Options As ServerOptions
```

[C#]
```
public HttpServer.ServerOptions Options {get; set;}
```

Property Value

A ServerOptions enumeration that specifies one or more server options.

Remarks

The **Options** property is used to specify one or more server options as bitflags using the ServerOptions enumeration. Each option has a corresponding property, and it is recommended that you use those properties, such as **LocalUser** and **UnixMode**, to specify whether a particular server option should be enabled or disabled. Using the class properties will make your code more readable and ensure forward compatibility.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.Priority Property

Gets and sets a value which specifies the server priority.

[Visual Basic]
```
Public Property Priority As ServerPriority
```

[C#]
```
public HttpServer.ServerPriority Priority {get; set;}
```

Property Value

Returns a ServerPriority enumeration value which specifies the current server priority. The default value for this property is **priorityNormal**.

Remarks

The **Priority** property can be used to control the processor usage, memory and network bandwidth allocated by the server for client sessions. The default priority balances resource utilization and client throughput while ensuring that the user interface remains responsive to the user. Lower priorities reduce the overall resource utilization at the expense of throughput.

Higher priority values increases the thread priority and processor utilization for the client sessions. It is not recommended that you increase the server priority unless you understand the implications of doing so and have thoroughly tested your application. Raising the priority of the server can have a negative impact on the responsiveness of the user interface.

See Also

HttpServer Class | SocketTools Namespace | ServerPriority Enumeration

HttpServer.ReadOnly Property

Determine if the server should prevent clients from uploading files.

[Visual Basic]

```
Public Property ReadOnly As Boolean
```

[C#]

```
public bool ReadOnly {get; set;}
```

Property Value

A Boolean value that specifies if clients have read-only access to the server.

Remarks

The **ReadOnly** property determines if the server should only allow read-only access to files by default, changing the default permissions granted to authenticated users. If this property is set to **True**, anonymous users will not be able to upload, rename or delete files and cannot create subdirectories. This is recommended if the server is publicly accessible over the Internet and guest logins are permitted.

Attempting to change the value of this property after the server has started will cause an exception to be raised. To change this property value, you must first call the **Stop** method which will terminate all active client connections.

The default value for this property is **False**.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.Restricted Property

Determine if the server should be started in restricted mode, limiting client access to the server.

[Visual Basic]
```
Public Property Restricted As Boolean
```

[C#]
```
public bool Restricted {get; set;}
```

Property Value

A boolean value that specifies if the server should be started in restricted mode. The default value for this property is **false**.

Remarks

The **Restricted** property determines if the server should be initialized in a restricted mode that isolates the server and limits the ability for clients to access files on the host system. If this property is set to **True**, the only commands accepted by the server will be the GET and HEAD commands. The server will never return a list of files if the client provides a URL that maps to a local directory and there is no default index page. Clients will not be able to execute CGI programs or scripts, and cannot access files outside of the server root directory or its subdirectories.

Attempting to change the value of this property after the server has started will cause an exception to be raised. To change this property value, you must first call the **Stop** method which will terminate all active client connections.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.Secure Property

Determine if the server should accept secure client connections.

[Visual Basic]
```
Public Property Secure As Boolean
```

[C#]
```
public bool Secure {get; set;}
```

Property Value

A Boolean value that specifies if the server should accept secure connections.

Remarks

The **Secure** property determines if client connections are encrypted using the standard SSL or TLS security protocols. The default value for this property is **False**, which specifies that clients will use a standard, unencrypted connection to the server. To enable secure connections, the application should set this property value to **True** prior to calling the **Start** method.

When secure connections are enabled, the server will accept the client connection and then wait for the client to initiate the handshake where both the client and server negotiate the various encryption options available. This process is handled automatically by the server, and all that is required is that the application specify the server certificate which should be used. This is done by setting the **CertificateName** property, and optionally the **CertificateStore** property if required.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.ServerAddress Property

Gets and sets the address that will be used by the server to listen for connections.

[Visual Basic]

```
Public Property ServerAddress As String
```

[C#]

```
public string ServerAddress {get; set;}
```

Property Value

A string which specifies the IP address that the server will use to listen for incoming client connections. An empty string indicates that the server will accept connections on any valid network interface configured for the local system.

Remarks

The **ServerAddress** property is used to specify the default address that the server will use when listening for connections. Setting this property to the value 0.0.0.0 or an empty string indicates that the server should listen for client connections using any valid network interface. If an address is specified, it must be a valid Internet address that is bound to a network adapter configured on the local system. Clients will only be able to connect to the server using that specific address.

It is common to set this property to the value 127.0.0.1 for testing purposes. It is a non-routable address that specifies the local system, and most software firewalls are configured so they do not block applications using this address.

See Also

HttpServer Class | SocketTools Namespace | ServerName Property | ServerPort Property

HttpServer.ServerName Property

Gets a value which specifies the host name for the local system.

[Visual Basic]

```
Public Property ServerName As String
```

[C#]

```
public string ServerName {get; set;}
```

Property Value

A string which specifies the local host name.

Remarks

The **ServerName** property returns the fully-qualified host name assigned to the local system. This consists of the local computer name and its domain name. The actual value returned depends on the system configuration. If no domain has been specified for the system, then only the machine name will be returned.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.ServerPort Property

Gets and sets the port number that will be used by the server to listen for connections**.**

[Visual Basic]

```
Public Property ServerPort As Integer
```

[C#]

```
public int ServerPort {get; set;}
```

Property Value

An integer value which specifies the port number.

Remarks

The **ServerPort** property is used to set the port number that server will use to listen for incoming client connections. Valid port numbers are in the range of 1 to 65535. It is recommended that most custom servers specify a port number larger than 5000 to avoid potential conflicts with standard Internet services and ephemeral ports used by client applications.

If a port number is specified that is already in use by another application, the **OnError** event will fire and the background server thread will terminate. To enable a server to be stopped and immediately restarted using the same address and port number, make sure that the **ReuseAddress** property is set to a value of **true**.

See Also

HttpServer Class | SocketTools Namespace | ServerAddress Property | ServerName Property

HttpServer.ServerThread Property

Gets the thread ID for the current server.

[Visual Basic]

```
Public ReadOnly Property ServerThread As Integer
```

[C#]

```
public int ServerThread {get;}
```

Property Value

An integer value which identifies the server thread that was created. A return value of zero specifies that no server has been started.

Remarks

Until the thread terminates, the thread identifier uniquely identifies the thread throughout the system.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.ServerUuid Property

Gets and sets the Universally Unique Identifier (UUID) associated with the server.

[Visual Basic]

```
Public Property ServerUuid As String
```

[C#]

```
public string ServerUuid {get; set;}
```

Property Value

A string that specifies the UUID assigned to the server instance.

Remarks

The **ServerUuid** property returns the UUID that uniquely identifies this instance of the server. If the application does not set this property, a temporary UUID will be assigned to the server. If a value is assigned to this property, it must be a valid UUID string. A permanent UUID can be generated using a utility such as **uuidgen** which is included with Visual Studio.

Attempting to change the value of this property after the server has started will cause an exception to be raised. To change this property value, you must first call the **Stop** method which will terminate all active client connections.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.StackSize Property

Gets and sets the size of the stack allocated for threads created by the server.

[Visual Basic]

```
Public Property StackSize As Integer
```

[C#]

```
public int StackSize {get; set;}
```

Property Value

An integer value that specifies the initial amount of memory that is committed to the stack for each thread created by the server.

Remarks

The default stack size for each thread is set to 256K for 32-bit processes and 512K for 64-bit processes. Increasing or decreasing the stack size will only affect new threads that are created by the server, it will not affect those threads that have already been created to manage active client sessions. It is recommended that most applications use the default stack size.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]

```
Public Property ThrowError As Boolean
```

[C#]

```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.Trace Property

Gets and sets a value which indicates if network function logging is enabled.

[Visual Basic]

```
Public Property Trace As Boolean
```

[C#]

```
public bool Trace {get; set;}
```

Property Value

Returns **true** if network function tracing is enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Trace** property is used to enable (or disable) the tracing of network function calls. When enabled, each function call is logged to a file, including the method parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the SocketTools networking classes and is enabled or disabled for an entire process. This means that once trace logging is enabled for a given component, all of the method calls made by the process using any of the SocketTools classes will be logged. For example, if you have an application using both the File Transfer Protocol and Post Office Protocol classes, and you set the **Trace** property to **true**, function calls made by both classes will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all classes used by the process.

If trace logging is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the logfile is fairly large. Since logfiles can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

When redistributing your application, make sure that you include the **SocketTools10.TraceLog.dll** module with your installation. If this library is not present, then no trace output will be generated and the value of the **Trace** property will be ignored. Only those function calls made by the SocketTools networking classes will be logged. Calls made directly to the Windows Sockets API, or calls made by other classes, will not be logged.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.TraceFile Property

Gets and sets a value which specifies the name of the network function tracing logfile.

[Visual Basic]

```
Public Property TraceFile As String
```

[C#]

```
public string TraceFile {get; set;}
```

Property Value

A string which specifies the name of the file.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named **SocketTools.log** is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFile** property should be set to the same value for each instance. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
 MyApp INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0 MyApp WRN:
connect(46, 192.0.0.1:1234, 16) returned -1 [10035] MyApp ERR: accept(46,
NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the method being called, the arguments passed to the method and the method's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the control is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Note that if the specified file cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.TraceFlags Property

Gets and sets a value which specifies the network function tracing flags.

[Visual Basic]

```
Public Property TraceFlags As TraceOptions
```

[C#]

```
public HttpServer.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration which specifies the amount of detail written to the trace logfile.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Because network function tracing is enabled per-process, the trace flags are shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFlags** property should be set to the same value for each instance. Changing the trace flags for any one instance of the class will affect the logging performed for all SocketTools classes used by the application.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being written and the error **errorOperationWouldBlock** occurs, a warning is generated because the application simply needs to attempt to write the data at a later time.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.Version Property

Gets a value which returns the current version of the HttpServer class library.

[Visual Basic]

```
Public ReadOnly Property Version As String
```

[C#]

```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the HttpServer class library. This value can be used by an application for validation and debugging purposes.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.VirtualPath Property

Return the virtual path to the local file or directory that is the target of the current command.

[Visual Basic]

```
Public Property VirtualPath As String
```

[C#]

```
public string VirtualPath {get; set;}
```

Property Value

A string that specifies the virtual path to the local file accessed by the active client session.

Remarks

The **VirtualPath** property returns the virtual path to a local file name or directory specified by the client as an argument to a standard HTTP command. For example, if the client sends the GET command to the server, this property will return the complete virtual path to the resource that the client wants to access. This property will only return a value for those standard commands that perform some action on a file or directory, otherwise it will return an empty string.

Setting this property allows you to effectively redirect the client to use a different file than the one that was actually requested. If the path is absolute, then it will be used as-is. If the path is relative, it will be relative to the current working directory for the active client session. If this property is set to an empty string, then the server will revert to using the actual file or directory name specified by the command.

This property should only be set within an **OnCommand** event handler, and only for those commands that perform an action on a file or directory. If the current command does not target a file or directory, setting this property will cause an exception to be raised by the control. Exercise caution when using this property to redirect the server to use a different file than the one requested by the client; changing the target file may cause the client to behave in unexpected ways.

See Also

HttpServer Class | SocketTools Namespace

HttpServer Methods

The methods of the **HttpServer** class are listed below. For a complete list of **HttpServer** class members, see the HttpServer Members topic.

Public Static (Shared) Methods

ErrorText	Returns the description of an error code.

Public Instance Methods

AddHost	Overloaded. Add a new virtual host to the server virtual host table.
AddPath	Overloaded. Add a new virtual path for the specified host.
AddUser	Overloaded. Add a new virtual user to the specified host.
Authenticate	Overloaded. Authenticate the client and assign access rights for the session.
CheckPath	Overloaded. Determine if the client has permission to access the specified virtual path.
ClearHeaders	Overloaded. Delete all of the response headers for the specified client session.
DeleteHost	Overloaded. Delete a virtual host associated with the specified server.
DeletePath	Overloaded. Delete a virtual path from the specified virtual host.
DeleteUser	Overloaded. Remove a virtual user from the server.
Disconnect	Overloaded. Disconnect the specified client session from the server.
Dispose	Overloaded. Releases all resources used by HttpServer.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetAllHeaders	Overloaded. Return all of the request header values in the specified string buffer.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetHeader	Overloaded. Return the value of a request header for the specified client session.
GetType (inherited from Object)	Gets the Type of the current instance.
GetVariable	Overloaded. Return the value of a CGI environment variable for the specified client.

Initialize	Overloaded. Initialize an instance of the HttpServer class.
ReceiveRequest	Overloaded. Receive the request that was sent by the client to the server.
RedirectRequest	Overloaded. Redirect the request from the client to another URL.
RegisterHandler	Overloaded. Register a CGI program for use and associate it with a file name extension.
RegisterProgram	Overloaded. Register a CGI program for use and associate it with a virtual path on the server.
RequireAuthentication	Overloaded. Send a response to the client indicating that authentication is required.
Reset	Reset the internal state of the object, resetting all properties to their default values.
ResolvePath	Overloaded. Resolve a path to its full virtual or local file name.
Restart	Restarts the server and terminates all active client connections.
Resume	Resume accepting new client connections.
SendError	Overloaded. Send an error result code and message to the client in response to a command.
SendResponse	Overloaded. Send a result code and message to the client in response to a command.
SetHeader	Overloaded. Create or change the value of a response header for the client session.
SetVariable	Overloaded. Create or change the value of a CGI environment variable for the specified client.
Start	Overloaded. Start listening for client connections on the specified IP address and port number.
Stop	Stop listening for new client connections and terminate all active clients already connected to the server.
Suspend	Suspend accepting new client connections.
Throttle	Overloaded. Limit the maximum number of client connections, connections per IP address and connection rate.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the server.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources

	allocated by the HttpServer class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.AddHost Method

Add a new virtual host to the server virtual host table.

Overload List

Add a new virtual host to the server virtual host table.

public int AddHost(string);

Add a new virtual host to the server virtual host table.

public int AddHost(string,string);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.AddHost Method (String, String)

Add a new virtual host to the server virtual host table.

[Visual Basic]

```
Overloads Public Function AddHost( _
   ByVal hostName As String, _
   ByVal rootDirectory As String _
) As Integer
```

[C#]

```
public int AddHost(
   string hostName,
   string rootDirectory
);
```

Parameters

hostName
A string which specifies the hostname that will be added to the virtual host table. This parameter must specify a valid hostname and cannot be a zero-length string.

rootDirectory
An optional string that specifies the root document directory for the virtual host. If this parameter is omitted or a zero-length string, the virtual host will use the same root directory that was specified when the server was started. This parameter may contain environment variables enclosed in % symbols.

Return Value

An integer value that uniquely identifies the virtual host that was added to the server. This value may be used with other methods that require a virtual host identifier. If the method fails, the return value will be -1 and the **LastError** property will be updated to indicate the cause of the failure.

Remarks

Virtual hosting is a method for sharing multiple domain names on a single instance of a server. The client provides the server with the hostname that it has used to establish the connection, and that name is compared against a table of virtual hosts configured for the server. If the hostname matches a virtual host, the client will use the root directory and any virtual paths that have been assigned to that host.

When the server is first started, a default virtual host with an ID of zero is automatically created and is identified as **httpHostDefault**. This virtual host uses the same hostname, port number and root directory that the server instance was created with. The application should treat all other host IDs as opaque values and never make assumptions about how they are allocated.

The virtual host ID returned by this method can be used with the **AddPath** method to create a virtual path assigned to the host, the **AddUser** to create a virtual user, and the **RegisterHandler** and **RegisterProgram** methods which are used to register script handlers and CGI programs.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.AddHost Overload List

HttpServer.AddHost Method (String)

Add a new virtual host to the server virtual host table.

[Visual Basic]

```
Overloads Public Function AddHost( _
   ByVal hostName As String _
) As Integer
```

[C#]

```
public int AddHost(
   string hostName
);
```

Parameters

hostName

A string which specifies the hostname that will be added to the virtual host table. This parameter must specify a valid hostname and cannot be a zero-length string.

Return Value

An integer value that uniquely identifies the virtual host that was added to the server. This value may be used with other methods that require a virtual host identifier. If the method fails, the return value will be -1 and the **LastError** property will be updated to indicate the cause of the failure.

Remarks

Virtual hosting is a method for sharing multiple domain names on a single instance of a server. The client provides the server with the hostname that it has used to establish the connection, and that name is compared against a table of virtual hosts configured for the server. If the hostname matches a virtual host, the client will use the root directory and any virtual paths that have been assigned to that host.

When the server is first started, a default virtual host with an ID of zero is automatically created and is identified as **httpHostDefault**. This virtual host uses the same hostname, port number and root directory that the server instance was created with. The application should treat all other host IDs as opaque values and never make assumptions about how they are allocated.

The virtual host ID returned by this method can be used with the **AddPath** method to create a virtual path assigned to the host, the **AddUser** to create a virtual user, and the **RegisterHandler** and **RegisterProgram** methods which are used to register script handlers and CGI programs.

This version of the method will share the same root directory that was specified when this instance of the server was started.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.AddHost Overload List

HttpServer.AddPath Method

Add a new virtual path for the specified host.

Overload List

Add a new virtual path for the specified host.

public bool AddPath(int,string,string);

Add a new virtual path for the specified host.

public bool AddPath(int,string,string,FileAccess);

Add a new virtual path for the default host.

public bool AddPath(string,string);

Add a new virtual path for the default host.

public bool AddPath(string,string,FileAccess);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.AddPath Method (Int32, String, String, FileAccess)

Add a new virtual path for the specified host.

[Visual Basic]

```
Overloads Public Function AddPath( _
   ByVal hostId As Integer, _
   ByVal virtualPath As String, _
   ByVal localPath As String, _
   ByVal accessFlags As FileAccess _
) As Boolean
```

[C#]

```
public bool AddPath(
   int hostId,
   string virtualPath,
   string localPath,
   FileAccess accessFlags
);
```

Parameters

hostId
: An integer value which identifies the virtual host. A value of zero specifies that the default virtual host should be used.

virtualPath
: A string which specifies the virtual path that will be created. This parameter cannot be an empty string and the maximum length of the virtual path is 1024 characters.

localPath
: A string which specifies the local directory or file name that the virtual path will be mapped to. This path must exist and can be no longer than 260 characters. This parameter cannot be an empty string.

accessFlags
: One or more FileAccess enumeration values which designates the access clients will be given to the virtual path.

Return Value

A boolean value which specifies if the virtual path was added to the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **AddPath** method maps a virtual path name to a directory or file name on the local system. Virtual paths are assigned to specific hosts and if multiple virtual hosts are created for the server, each can have its own virtual paths which map to different files. To create a virtual path for the default server, the caller should specify the *HostId* parameter as **HttpServer.hostDefault** which has a value of zero.

It is recommended that the *localPath* parameter always specify the full path to the local file or directory. If the path is relative, it will be considered relative to the current working directory for the process and expanded to its full path name. The local path can include environment variables surrounded by % symbols. For example, if the value %ProgramData% is included in the path, it will be expanded to the full path for the common application data folder. The local path cannot specify a Windows system folder or the root directory of a mounted drive volume.

The local file or directory does not need to located in the document root directory for the server or virtual host. It can specify any valid local path that the server process has the appropriate permissions to access. You should exercise caution when creating virtual paths to files or directories outside of the server root directory. If the ***LocalPath*** parameter specifies a directory, clients will have access to that directory and all subdirectories using its virtual path.

If you wish to password protect the virtual file or directory, include the **accessProtected** flag in the file permissions. The default command handlers will recognize this flag and require that the client authenticate itself to grant access to the resource. If the server application implements a custom command handler, it is responsible for checking for the presence of this flag and perform the appropriate checks to ensure that the client session has been authenticated.

If the server was started in restricted mode, the client will be unable to access documents outside of the server root directory and its subdirectories. This restriction also applies to virtual paths that reference documents or other resources outside of the root directory. To allow a client to access a document outside of the server root directory, the **ClientAccess** property should be used to grant the client **accessRead** permission.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.AddPath Overload List

HttpServer.AddPath Method (Int32, String, String)

Add a new virtual path for the specified host.

[Visual Basic]

```
Overloads Public Function AddPath( _
   ByVal hostId As Integer, _
   ByVal virtualPath As String, _
   ByVal localPath As String _
) As Boolean
```

[C#]

```
public bool AddPath(
   int hostId,
   string virtualPath,
   string localPath
);
```

Parameters

hostId
An integer value which identifies the virtual host. A value of zero specifies that the default virtual host should be used.

virtualPath
A string which specifies the virtual path that will be created. This parameter cannot be an empty string and the maximum length of the virtual path is 1024 characters.

localPath
A string which specifies the local directory or file name that the virtual path will be mapped to. This path must exist and can be no longer than 260 characters. This parameter cannot be an empty string.

Return Value

A boolean value which specifies if the virtual path was added to the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **AddPath** method maps a virtual path name to a directory or file name on the local system. Virtual paths are assigned to specific hosts and if multiple virtual hosts are created for the server, each can have its own virtual paths which map to different files. To create a virtual path for the default server, the caller should specify the ***HostId*** parameter as **HttpServer.hostDefault** which has a value of zero.

It is recommended that the *localPath* parameter always specify the full path to the local file or directory. If the path is relative, it will be considered relative to the current working directory for the process and expanded to its full path name. The local path can include environment variables surrounded by % symbols. For example, if the value %ProgramData% is included in the path, it will be expanded to the full path for the common application data folder. The local path cannot specify a Windows system folder or the root directory of a mounted drive volume.

The local file or directory does not need to located in the document root directory for the server or virtual host. It can specify any valid local path that the server process has the appropriate permissions to access. You should exercise caution when creating virtual paths to files or directories outside of the server root directory. If the *LocalPath* parameter specifies a directory, clients will have access to that directory and all subdirectories using its virtual path.

If you wish to password protect the virtual file or directory, include the **accessProtected** flag in the file permissions. The default command handlers will recognize this flag and require that the client authenticate itself to grant access to the resource. If the server application implements a custom command handler, it is responsible for checking for the presence of this flag and perform the appropriate checks to ensure that the client session has been authenticated.

If the server was started in restricted mode, the client will be unable to access documents outside of the server root directory and its subdirectories. This restriction also applies to virtual paths that reference documents or other resources outside of the root directory. To allow a client to access a document outside of the server root directory, the **ClientAccess** property should be used to grant the client **accessRead** permission.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.AddPath Overload List

HttpServer.AddPath Method (String, String, FileAccess)

Add a new virtual path for the default host.

[Visual Basic]

```
Overloads Public Function AddPath( _
   ByVal virtualPath As String, _
   ByVal localPath As String, _
   ByVal accessFlags As FileAccess _
) As Boolean
```

[C#]

```
public bool AddPath(
   string virtualPath,
   string localPath,
   FileAccess accessFlags
);
```

Parameters

virtualPath

A string which specifies the virtual path that will be created. This parameter cannot be an empty string and the maximum length of the virtual path is 1024 characters.

localPath

A string which specifies the local directory or file name that the virtual path will be mapped to. This path must exist and can be no longer than 260 characters. This parameter cannot be an empty string.

accessFlags

One or more FileAccess enumeration values which designates the access clients will be given to the virtual path.

Return Value

A boolean value which specifies if the virtual path was added to the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **AddPath** method maps a virtual path name to a directory or file name on the local system. Virtual paths are assigned to specific hosts and if multiple virtual hosts are created for the server, each can have its own virtual paths which map to different files. To create a virtual path for the default server, the caller should specify the ***HostId*** parameter as **HttpServer.hostDefault** which has a value of zero.

It is recommended that the ***localPath*** parameter always specify the full path to the local file or directory. If the path is relative, it will be considered relative to the current working directory for the process and expanded to its full path name. The local path can include environment variables surrounded by % symbols. For example, if the value %ProgramData% is included in the path, it will be expanded to the full path for the common application data folder. The local path cannot specify a Windows system folder or the root directory of a mounted drive volume.

The local file or directory does not need to located in the document root directory for the server or virtual host. It can specify any valid local path that the server process has the appropriate permissions to access. You should exercise caution when creating virtual paths to files or directories outside of the server root directory. If the ***LocalPath*** parameter specifies a directory, clients will have access to that directory and all subdirectories using its virtual path.

If you wish to password protect the virtual file or directory, include the **accessProtected** flag in the file permissions. The default command handlers will recognize this flag and require that the client authenticate itself to grant access to the resource. If the server application implements a custom command handler, it is responsible for checking for the presence of this flag and perform the appropriate checks to ensure that the client session has been authenticated.

If the server was started in restricted mode, the client will be unable to access documents outside of the server root directory and its subdirectories. This restriction also applies to virtual paths that reference documents or other resources outside of the root directory. To allow a client to access a document outside of the server root directory, the **ClientAccess** property should be used to grant the client **accessRead** permission.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.AddPath Overload List

HttpServer.AddPath Method (String, String)

Add a new virtual path for the default host.

```
[Visual Basic]
Overloads Public Function AddPath( _
   ByVal virtualPath As String, _
   ByVal localPath As String _
) As Boolean
```

```
[C#]
public bool AddPath(
   string virtualPath,
   string localPath
);
```

Parameters

virtualPath
A string which specifies the virtual path that will be created. This parameter cannot be an empty string and the maximum length of the virtual path is 1024 characters.

localPath
A string which specifies the local directory or file name that the virtual path will be mapped to. This path must exist and can be no longer than 260 characters. This parameter cannot be an empty string.

Return Value

A boolean value which specifies if the virtual path was added to the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **AddPath** method maps a virtual path name to a directory or file name on the local system. Virtual paths are assigned to specific hosts and if multiple virtual hosts are created for the server, each can have its own virtual paths which map to different files. To create a virtual path for the default server, the caller should specify the ***HostId*** parameter as **HttpServer.hostDefault** which has a value of zero.

It is recommended that the *localPath* parameter always specify the full path to the local file or directory. If the path is relative, it will be considered relative to the current working directory for the process and expanded to its full path name. The local path can include environment variables surrounded by % symbols. For example, if the value %ProgramData% is included in the path, it will be expanded to the full path for the common application data folder. The local path cannot specify a Windows system folder or the root directory of a mounted drive volume.

The local file or directory does not need to located in the document root directory for the server or virtual host. It can specify any valid local path that the server process has the appropriate permissions to access. You should exercise caution when creating virtual paths to files or directories outside of the server root directory. If the *LocalPath* parameter specifies a directory, clients will have access to that directory and all subdirectories using its virtual path.

If you wish to password protect the virtual file or directory, include the **accessProtected** flag in the file permissions. The default command handlers will recognize this flag and require that the client authenticate itself to grant access to the resource. If the server application implements a custom command handler, it is responsible for checking for the presence of this flag and perform the appropriate checks to ensure that the client session has been authenticated.

If the server was started in restricted mode, the client will be unable to access documents outside of the server root directory and its subdirectories. This restriction also applies to virtual paths that reference documents or other resources outside of the root directory. To allow a client to access a document outside of the server root directory, the **ClientAccess** property should be used to grant the client **accessRead** permission.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.AddPath Overload List

HttpServer.AddUser Method

Add a new virtual user to the specified host.

Overload List

Add a new virtual user to the specified host.

public bool AddUser(int,string,string);

Add a new virtual user to the specified host.

public bool AddUser(int,string,string,UserAccess,string);

Add a new virtual user to the specified host.

public bool AddUser(int,string,string,string);

Add a new virtual user to the default host.

public bool AddUser(string,string);

Add a new virtual user to the default host.

public bool AddUser(string,string,UserAccess,string);

Add a new virtual user to the default host.

public bool AddUser(string,string,string);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.AddUser Method (String, String)

Add a new virtual user to the default host.

[Visual Basic]

```
Overloads Public Function AddUser( _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]

```
public bool AddUser(
   string userName,
   string userPassword
);
```

Parameters

userName

A string which specifies the user name. The maximum length of a username is 63 characters and it is recommended that names be limited to alphanumeric characters. Whitespace, control characters and certain symbols such as path delimiters and wildcard characters are not permitted. If an invalid character is included in the name, the method will fail with an error indicating the username is invalid. The username must be at least three characters in length. Usernames are not case sensitive.

userPassword

A string which specifies the user password. The maximum length of a password is 63 characters and is limited to printable characters. Whitespace and control characters are not permitted. If an invalid character is included in the password, the method will fail with an error indicating the password is invalid. The password must be at least one character in length. Passwords are case sensitive.

Return Value

A boolean value which specifies if the virtual user was added to the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **AddUser** method creates a virtual user that is associated with the server. When a client connects with the server and provides authentication credentials, the server will check if the username has been created using this method. If a match is found, the client access rights will be updated.

If you wish to modify the information for an existing user, it is not necessary to delete the username first. If this method is called with a username that already exists, that record is replaced with the values passed to this method.

The virtual users created by this method exist only as long as the server is active. If you wish to maintain a persistent database of users and passwords, you are responsible for its implementation based on the requirements of your specific application. For example, a simple implementation would be to store the user information in a local XML or INI file and then read that configuration file after the server has started, calling this method for each user that is listed.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.AddUser Overload List

HttpServer.AddUser Method (String, String, String)

Add a new virtual user to the default host.

[Visual Basic]

```
Overloads Public Function AddUser( _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal homeDirectory As String _
) As Boolean
```

[C#]

```
public bool AddUser(
   string userName,
   string userPassword,
   string homeDirectory
);
```

Parameters

userName

A string which specifies the user name. The maximum length of a username is 63 characters and it is recommended that names be limited to alphanumeric characters. Whitespace, control characters and certain symbols such as path delimiters and wildcard characters are not permitted. If an invalid character is included in the name, the method will fail with an error indicating the username is invalid. The username must be at least three characters in length. Usernames are not case sensitive.

userPassword

A string which specifies the user password. The maximum length of a password is 63 characters and is limited to printable characters. Whitespace and control characters are not permitted. If an invalid character is included in the password, the method will fail with an error indicating the password is invalid. The password must be at least one character in length. Passwords are case sensitive.

homeDirectory

A string which specifies the directory that will be the virtual user's home directory. If the server was started in multi-user mode, this directory will be relative to the user directory created by the server, otherwise it will be relative to the server root directory. If the directory does not exist, it will be created the first time that the virtual user successfully logs in to the server. If this parameter is an empty string, a default home directory will be created for the virtual user.

Return Value

A boolean value which specifies if the virtual user was added to the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **AddUser** method creates a virtual user that is associated with the server. When a client connects with the server and provides authentication credentials, the server will check if the username has been created using this method. If a match is found, the client access rights will be updated.

If you wish to modify the information for an existing user, it is not necessary to delete the username first. If this method is called with a username that already exists, that record is replaced with the values passed to this method.

The virtual users created by this method exist only as long as the server is active. If you wish to maintain a persistent database of users and passwords, you are responsible for its implementation based on the requirements of your specific application. For example, a simple implementation would be to store the user information in a local XML or INI file and then read that configuration file after the server has started, calling this method for each user that is listed.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.AddUser Overload List

HttpServer.AddUser Method (String, String, UserAccess, String)

Add a new virtual user to the default host.

[Visual Basic]

```
Overloads Public Function AddUser( _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal accessFlags As UserAccess, _
   ByVal homeDirectory As String _
) As Boolean
```

[C#]

```
public bool AddUser(
   string userName,
   string userPassword,
   UserAccess accessFlags,
   string homeDirectory
);
```

Parameters

userName

A string which specifies the user name. The maximum length of a username is 63 characters and it is recommended that names be limited to alphanumeric characters. Whitespace, control characters and certain symbols such as path delimiters and wildcard characters are not permitted. If an invalid character is included in the name, the method will fail with an error indicating the username is invalid. The username must be at least three characters in length. Usernames are not case sensitive.

userPassword

A string which specifies the user password. The maximum length of a password is 63 characters and is limited to printable characters. Whitespace and control characters are not permitted. If an invalid character is included in the password, the method will fail with an error indicating the password is invalid. The password must be at least one character in length. Passwords are case sensitive.

accessFlags

A UserAccess enumeration which specifies the access clients will be given when authenticated as this user.

homeDirectory

A string which specifies the directory that will be the virtual user's home directory. If the server was started in multi-user mode, this directory will be relative to the user directory created by the server, otherwise it will be relative to the server root directory. If the directory does not exist, it will be created the first time that the virtual user successfully logs in to the server. If this parameter is an empty string, a default home directory will be created for the virtual user.

Return Value

A boolean value which specifies if the virtual user was added to the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **AddUser** method creates a virtual user that is associated with the server. When a client connects with the server and provides authentication credentials, the server will check if the username has been created using this method. If a match is found, the client access rights will be updated.

If you wish to modify the information for an existing user, it is not necessary to delete the username first. If this method is called with a username that already exists, that record is replaced with the values passed to this method.

The virtual users created by this method exist only as long as the server is active. If you wish to maintain a persistent database of users and passwords, you are responsible for its implementation based on the requirements of your specific application. For example, a simple implementation would be to store the user information in a local XML or INI file and then read that configuration file after the server has started, calling this method for each user that is listed.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.AddUser Overload List

HttpServer.AddUser Method (Int32, String, String)

Add a new virtual user to the specified host.

[Visual Basic]

```
Overloads Public Function AddUser( _
   ByVal hostId As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]

```
public bool AddUser(
   int hostId,
   string userName,
   string userPassword
);
```

Parameters

hostId

An integer value which identifies the virtual host. A value of zero specifies that the default virtual host should be used.

userName

A string which specifies the user name. The maximum length of a username is 63 characters and it is recommended that names be limited to alphanumeric characters. Whitespace, control characters and certain symbols such as path delimiters and wildcard characters are not permitted. If an invalid character is included in the name, the method will fail with an error indicating the username is invalid. The username must be at least three characters in length. Usernames are not case sensitive.

userPassword

A string which specifies the user password. The maximum length of a password is 63 characters and is limited to printable characters. Whitespace and control characters are not permitted. If an invalid character is included in the password, the method will fail with an error indicating the password is invalid. The password must be at least one character in length. Passwords are case sensitive.

Return Value

A boolean value which specifies if the virtual user was added to the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **AddUser** method creates a virtual user that is associated with the server. When a client connects with the server and provides authentication credentials, the server will check if the username has been created using this method. If a match is found, the client access rights will be updated.

If you wish to modify the information for an existing user, it is not necessary to delete the username first. If this method is called with a username that already exists, that record is replaced with the values passed to this method.

The virtual users created by this method exist only as long as the server is active. If you wish to maintain a persistent database of users and passwords, you are responsible for its implementation based on the requirements of your specific application. For example, a simple implementation would be to store the user information in a local XML or INI file and then read that configuration file after the server has started, calling this method for each user that is listed.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.AddUser Overload List

HttpServer.AddUser Method (Int32, String, String, String)

Add a new virtual user to the specified host.

[Visual Basic]

```
Overloads Public Function AddUser( _
   ByVal hostId As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal homeDirectory As String _
) As Boolean
```

[C#]

```
public bool AddUser(
   int hostId,
   string userName,
   string userPassword,
   string homeDirectory
);
```

Parameters

hostId

An integer value which identifies the virtual host. A value of zero specifies that the default virtual host should be used.

userName

A string which specifies the user name. The maximum length of a username is 63 characters and it is recommended that names be limited to alphanumeric characters. Whitespace, control characters and certain symbols such as path delimiters and wildcard characters are not permitted. If an invalid character is included in the name, the method will fail with an error indicating the username is invalid. The username must be at least three characters in length. Usernames are not case sensitive.

userPassword

A string which specifies the user password. The maximum length of a password is 63 characters and is limited to printable characters. Whitespace and control characters are not permitted. If an invalid character is included in the password, the method will fail with an error indicating the password is invalid. The password must be at least one character in length. Passwords are case sensitive.

homeDirectory

A string which specifies the directory that will be the virtual user's home directory. If the server was started in multi-user mode, this directory will be relative to the user directory created by the server, otherwise it will be relative to the server root directory. If the directory does not exist, it will be created the first time that the virtual user successfully logs in to the server. If this parameter is an empty string, a default home directory will be created for the virtual user.

Return Value

A boolean value which specifies if the virtual user was added to the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **AddUser** method creates a virtual user that is associated with the server. When a client connects with the server and provides authentication credentials, the server will check if the username has been created using this method. If a match is found, the client access rights will be updated.

If you wish to modify the information for an existing user, it is not necessary to delete the username first. If this method is called with a username that already exists, that record is replaced with the values passed to this method.

The virtual users created by this method exist only as long as the server is active. If you wish to maintain a persistent database of users and passwords, you are responsible for its implementation based on the requirements of your specific application. For example, a simple implementation would be to store the user information in a local XML or INI file and then read that configuration file after the server has started, calling this method for each user that is listed.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.AddUser Overload List

HttpServer.AddUser Method (Int32, String, String, UserAccess, String)

Add a new virtual user to the specified host.

[Visual Basic]

```
Overloads Public Function AddUser( _
   ByVal hostId As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal accessFlags As UserAccess, _
   ByVal homeDirectory As String _
) As Boolean
```

[C#]

```
public bool AddUser(
   int hostId,
   string userName,
   string userPassword,
   UserAccess accessFlags,
   string homeDirectory
);
```

Parameters

hostId

An integer value which identifies the virtual host. A value of zero specifies that the default virtual host should be used.

userName

A string which specifies the user name. The maximum length of a username is 63 characters and it is recommended that names be limited to alphanumeric characters. Whitespace, control characters and certain symbols such as path delimiters and wildcard characters are not permitted. If an invalid character is included in the name, the method will fail with an error indicating the username is invalid. The username must be at least three characters in length. Usernames are not case sensitive.

userPassword

A string which specifies the user password. The maximum length of a password is 63 characters and is limited to printable characters. Whitespace and control characters are not permitted. If an invalid character is included in the password, the method will fail with an error indicating the password is invalid. The password must be at least one character in length. Passwords are case sensitive.

accessFlags

A UserAccess enumeration which specifies the access clients will be given when authenticated as this user.

homeDirectory

A string which specifies the directory that will be the virtual user's home directory. If the server was started in multi-user mode, this directory will be relative to the user directory created by the server, otherwise it will be relative to the server root directory. If the directory does not exist, it will be created the first time that the virtual user successfully logs in to the server. If this parameter is an empty string, a default home directory will be created for the virtual user.

Return Value

A boolean value which specifies if the virtual user was added to the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **AddUser** method creates a virtual user that is associated with the server. When a client connects with the server and provides authentication credentials, the server will check if the username has been created using this method. If a match is found, the client access rights will be updated.

If you wish to modify the information for an existing user, it is not necessary to delete the username first. If this method is called with a username that already exists, that record is replaced with the values passed to this method.

The virtual users created by this method exist only as long as the server is active. If you wish to maintain a persistent database of users and passwords, you are responsible for its implementation based on the requirements of your specific application. For example, a simple implementation would be to store the user information in a local XML or INI file and then read that configuration file after the server has started, calling this method for each user that is listed.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.AddUser Overload List

HttpServer.Authenticate Method

Authenticate the active client and assign access rights for the session.

Overload List

Authenticate the active client and assign access rights for the session.

public bool Authenticate(UserAccess);

Authenticate the client and assign access rights for the session.

public bool Authenticate(int,UserAccess);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.Authenticate Method (Int32, UserAccess)

Authenticate the client and assign access rights for the session.

[Visual Basic]

```
Overloads Public Function Authenticate( _
   ByVal clientId As Integer, _
   ByVal accessFlags As UserAccess _
) As Boolean
```

[C#]

```
public bool Authenticate(
   int clientId,
   UserAccess accessFlags
);
```

Parameters

clientId
: An integer that identifies the client session.

accessFlags
: One or more UserAccess enumeration values that specifies the access clients will be given when authenticated as this user.

Return Value

A boolean value which specifies if the client session was authenticated. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Authenticate** method is used to authenticate a specific client session, typically in response to an **OnAuthenticate** event that indicates a client has provided authentication credentials as part of the request for a document or other resource.

To enable the server to automatically authenticate a client session, use the **AddUser** method to add one or more virtual users. The server will search the list of virtual users for a match to the credentials provided by the client and will set the appropriate permissions for the session without requiring a event handler to manually authenticate the session using this method.

If the server was started with the **LocalUser** property set to **True** and the client session is not authenticated using this method, the server will attempt to authenticate the client session using the local Windows user database. Although this option can be convenient because it does not require the implementation of an event handler for the **OnAuthenticate** event, it can be used by clients to attempt to discover valid usernames and passwords for the local system. It is recommended that you use the **AddUser** method to create virtual users rather than using the local user database.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.Authenticate Overload List

HttpServer.Authenticate Method (UserAccess)

Authenticate the active client and assign access rights for the session.

[Visual Basic]

```
Overloads Public Function Authenticate( _
   ByVal accessFlags As UserAccess _
) As Boolean
```

[C#]

```
public bool Authenticate(
   UserAccess accessFlags
);
```

Parameters

accessFlags

One or more UserAccess enumeration values that specifies the access clients will be given when authenticated as this user.

Return Value

A boolean value which specifies if the client session was authenticated. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Authenticate** method is used to authenticate the active client session, typically in response to an **OnAuthenticate** event that indicates a client has provided authentication credentials as part of the request for a document or other resource.

To enable the server to automatically authenticate a client session, use the **AddUser** method to add one or more virtual users. The server will search the list of virtual users for a match to the credentials provided by the client and will set the appropriate permissions for the session without requiring a event handler to manually authenticate the session using this method.

If the server was started with the **LocalUser** property set to **True** and the client session is not authenticated using this method, the server will attempt to authenticate the client session using the local Windows user database. Although this option can be convenient because it does not require the implementation of an event handler for the **OnAuthenticate** event, it can be used by clients to attempt to discover valid usernames and passwords for the local system. It is recommended that you use the **AddUser** method to create virtual users rather than using the local user database.

This version of the method uses the active client session and should only be called from within a server event handler. To specify a client session outside of an event handler, use the version of this method that accepts a client ID parameter.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.Authenticate Overload List

HttpServer.CheckPath Method

Determine if the client has read access to the specified virtual path.

Overload List

Determine if the client has read access to the specified virtual path.

public bool CheckPath(int,string);

Determine if the client has permission to access the specified virtual path.

public bool CheckPath(int,string,FileAccess);

Determine if the active client has read access to the specified virtual path.

public bool CheckPath(string);

Determine if the active client has permission to access the specified virtual path.

public bool CheckPath(string,FileAccess);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.CheckPath Method (String)

Determine if the active client has read access to the specified virtual path.

[Visual Basic]

```
Overloads Public Function CheckPath( _
   ByVal virtualPath As String _
) As Boolean
```

[C#]

```
public bool CheckPath(
   string virtualPath
);
```

Parameters

virtualPath

A string which specifies the virtual path that will be created. The path must be absolute and cannot be an empty string. The maximum length of the virtual path is 1024 characters.

Return Value

A boolean value which specifies if the virtual path can be accessed by the client. A return value of **true** specifies that the file or directory can be accessed. If the file or directory cannot be accessed, the method returns false and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **CheckPath** method is used to determine if the client has permission to read the virtual file or directory. The method will return a non-zero value if the client does have the requested permission, or zero if it does not.

Applications that implement their own custom handlers for standard HTTP commands should use this method to ensure that the client has the appropriate permissions to access the requested resource. Failure to check the access permissions for the client can result in the client being able to access restricted documents and other resources. It is recommended that most applications use the default command handlers.

This version of the method uses the active client session and should only be called from within a server event handler. To specify a client session outside of an event handler, use the version of this method that accepts a client ID parameter.

To obtain the path to the local file or directory that the virtual path is mapped to, use the **ResolvePath** method.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.CheckPath Overload List

HttpServer.CheckPath Method (String, FileAccess)

Determine if the active client has permission to access the specified virtual path.

[Visual Basic]

```
Overloads Public Function CheckPath( _
   ByVal virtualPath As String, _
   ByVal accessFlags As FileAccess _
) As Boolean
```

[C#]

```
public bool CheckPath(
   string virtualPath,
   FileAccess accessFlags
);
```

Parameters

virtualPath
A string which specifies the virtual path that will be created. The path must be absolute and cannot be an empty string. The maximum length of the virtual path is 1024 characters.

accessFlags
One or more FileAccess enumerations that specifies the access permissions that should be checked.

Return Value

A boolean value which specifies if the virtual path can be accessed by the client. A return value of **true** specifies that the file or directory can be accessed with the requested permissions. If the file or directory cannot be accessed, the method returns false and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **CheckPath** method is used to determine if the client has permission to access the virtual file or directory, based on the value of the ***accessFlags*** parameter. For example, if the ***accessFlags*** parameter has the value **accessWrite**, this method will check if the client has write permission for the file or directory. The method will return a non-zero value if the client does have the requested permission, or zero if it does not.

Applications that implement their own custom handlers for standard HTTP commands should use this method to ensure that the client has the appropriate permissions to access the requested resource. Failure to check the access permissions for the client can result in the client being able to access restricted documents and other resources. It is recommended that most applications use the default command handlers.

This version of the method uses the active client session and should only be called from within a server event handler. To specify a client session outside of an event handler, use the version of this method that accepts a client ID parameter.

To obtain the path to the local file or directory that the virtual path is mapped to, use the **ResolvePath** method.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.CheckPath Overload List

HttpServer.CheckPath Method (Int32, String)

Determine if the client has read access to the specified virtual path.

[Visual Basic]
```
Overloads Public Function CheckPath( _
   ByVal clientId As Integer, _
   ByVal virtualPath As String _
) As Boolean
```

[C#]
```
public bool CheckPath(
   int clientId,
   string virtualPath
);
```

Parameters

clientId

An integer value which identifies the client session.

virtualPath

A string which specifies the virtual path that will be created. The path must be absolute and cannot be an empty string. The maximum length of the virtual path is 1024 characters.

Return Value

A boolean value which specifies if the virtual path can be accessed by the client. A return value of **true** specifies that the file or directory can be accessed. If the file or directory cannot be accessed, the method returns false and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **CheckPath** method is used to determine if the client has permission to read the virtual file or directory. The method will return a non-zero value if the client does have the requested permission, or zero if it does not.

Applications that implement their own custom handlers for standard HTTP commands should use this method to ensure that the client has the appropriate permissions to access the requested resource. Failure to check the access permissions for the client can result in the client being able to access restricted documents and other resources. It is recommended that most applications use the default command handlers.

To obtain the path to the local file or directory that the virtual path is mapped to, use the **ResolvePath** method.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.CheckPath Overload List

HttpServer.CheckPath Method (Int32, String, FileAccess)

Determine if the client has permission to access the specified virtual path.

[Visual Basic]

```
Overloads Public Function CheckPath( _
   ByVal clientId As Integer, _
   ByVal virtualPath As String, _
   ByVal accessFlags As FileAccess _
) As Boolean
```

[C#]

```
public bool CheckPath(
   int clientId,
   string virtualPath,
   FileAccess accessFlags
);
```

Parameters

clientId

An integer value which identifies the client session.

virtualPath

A string which specifies the virtual path that will be created. The path must be absolute and cannot be an empty string. The maximum length of the virtual path is 1024 characters.

accessFlags

One or more FileAccess enumerations that specifies the access permissions that should be checked.

Return Value

A boolean value which specifies if the virtual path can be accessed by the client. A return value of **true** specifies that the file or directory can be accessed with the requested permissions. If the file or directory cannot be accessed, the method returns false and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **CheckPath** method is used to determine if the client has permission to access the virtual file or directory, based on the value of the ***accessFlags*** parameter. For example, if the ***accessFlags*** parameter has the value **accessWrite**, this method will check if the client has write permission for the file or directory. The method will return a non-zero value if the client does have the requested permission, or zero if it does not.

Applications that implement their own custom handlers for standard HTTP commands should use this method to ensure that the client has the appropriate permissions to access the requested resource. Failure to check the access permissions for the client can result in the client being able to access restricted documents and other resources. It is recommended that most applications use the default command handlers.

To obtain the path to the local file or directory that the virtual path is mapped to, use the **ResolvePath** method.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.CheckPath Overload List

HttpServer.ClearHeaders Method

Delete all of the response headers for the active client session.

Overload List

Delete all of the response headers for the active client session.

public bool ClearHeaders();

Delete all of the response headers for the specified client session.

public bool ClearHeaders(int);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.ClearHeaders Method (Int32)

Delete all of the response headers for the specified client session.

[Visual Basic]

```
Overloads Public Function ClearHeaders( _
   ByVal clientId As Integer _
) As Boolean
```

[C#]

```
public bool ClearHeaders(
   int clientId
);
```

Parameters

clientId

An integer value which identifies the client session.

Return Value

A boolean value which specifies if the response headers were cleared. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **ClearHeaders** method is used to delete all of the current response header values and automatically generate a new set of default response headers. This method can be useful if the client application wants to clear any custom headers that were specified prior to sending a response to the client. In most cases it is not necessary to use this method because the server will automatically clear the response headers when a session terminates.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.ClearHeaders Overload List

HttpServer.ClearHeaders Method ()

Delete all of the response headers for the active client session.

[Visual Basic]

```
Overloads Public Function ClearHeaders() As Boolean
```

[C#]

```
public bool ClearHeaders();
```

Return Value

A boolean value which specifies if the response headers were cleared. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **ClearHeaders** method is used to delete all of the current response header values and automatically generate a new set of default response headers. This method can be useful if the client application wants to clear any custom headers that were specified prior to sending a response to the client. In most cases it is not necessary to use this method because the server will automatically clear the response headers when a session terminates.

This version of the method uses the active client session and should only be called from within a server event handler. To specify a client session outside of an event handler, use the version of this method that accepts a client ID parameter.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.ClearHeaders Overload List

HttpServer.DeleteHost Method

Delete a virtual host associated with the specified server.

Overload List

Delete a virtual host associated with the specified server.

public bool DeleteHost(int);

Delete a virtual host associated with the specified server.

public bool DeleteHost(string);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.DeleteHost Method (Int32)

Delete a virtual host associated with the specified server.

[Visual Basic]

```
Overloads Public Function DeleteHost( _
   ByVal hostId As Integer _
) As Boolean
```

[C#]

```
public bool DeleteHost(
   int hostId
);
```

Parameters

hostId

An integer value which identifies the virtual host. A value of zero specifies that the default virtual host should be used.

Return Value

A boolean value which specifies if the virtual host was deleted from the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **DeleteHost** method removes a virtual host that was created by a previous call to the **AddHost** method. All virtual paths and users associated with the specified host are no longer valid. It is not necessary to call this method to delete any of the virtual hosts prior to stopping the server. Part of the normal shutdown process is releasing the resources allocated for each virtual host that was added to the server.

This method cannot be used to delete the virtual host with an ID of zero, which is the default virtual host that is allocated when the server is started.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.DeleteHost Overload List

HttpServer.DeleteHost Method (String)

Delete a virtual host associated with the specified server.

[Visual Basic]
```
Overloads Public Function DeleteHost( _
   ByVal hostName As String _
) As Boolean
```

[C#]
```
public bool DeleteHost(
   string hostName
);
```

Parameters

hostName

A string that specifies the name of the virtual host that was previous added to the server. This value must match the complete virtual host name.

Return Value

A boolean value which specifies if the virtual host was removed from the server. A return value of true specifies that the operation was successful. If an error occurs, the method returns false and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **DeleteHost** method removes a virtual host that was created by a previous call to the **AddHost** method. All virtual paths and users associated with the specified host are no longer valid. It is not necessary to call this method to delete any of the virtual hosts prior to stopping the server. Part of the normal shutdown process is releasing the resources allocated for each virtual host that was added to the server.

This method cannot be used to delete the default virtual host that is created when the server is started.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.DeleteHost Overload List

HttpServer.DeletePath Method

Delete a virtual path from the specified virtual host.

Overload List

Delete a virtual path from the specified virtual host.

public bool DeletePath(int,string);

Delete a virtual path from the default virtual host.

public bool DeletePath(string);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.DeletePath Method (Int32, String)

Delete a virtual path from the specified virtual host.

[Visual Basic]

```
Overloads Public Function DeletePath( _
   ByVal hostId As Integer, _
   ByVal virtualPath As String _
) As Boolean
```

[C#]

```
public bool DeletePath(
   int hostId,
   string virtualPath
);
```

Parameters

hostId
An integer value which identifies the virtual host. A value of zero specifies that the default virtual host should be used.

virtualPath
A string that specifies the virtual path to be removed. This path must be absolute and cannot be an empty string.

Return Value

A boolean value which specifies if the virtual path was removed from the server. A return **value** of true specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.DeletePath Overload List

HttpServer.DeletePath Method (String)

Delete a virtual path from the default virtual host.

[Visual Basic]

```
Overloads Public Function DeletePath( _
   ByVal virtualPath As String _
) As Boolean
```

[C#]

```
public bool DeletePath(
   string virtualPath
);
```

Parameters

virtualPath

A string that specifies the virtual path to be removed. This path must be absolute and cannot be an empty string.

Return Value

A boolean value which specifies if the virtual path was removed from the server. A return **value** of true specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.DeletePath Overload List

HttpServer.DeleteUser Method

Remove a virtual user from the server.

Overload List

Remove a virtual user from the server.

public bool DeleteUser(int,string);

Remove a virtual user from the server.

public bool DeleteUser(string);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.DeleteUser Method (Int32, String)

Remove a virtual user from the server.

[Visual Basic]

```
Overloads Public Function DeleteUser( _
   ByVal hostId As Integer, _
   ByVal userName As String _
) As Boolean
```

[C#]

```
public bool DeleteUser(
   int hostId,
   string userName
);
```

Parameters

hostId

An integer value which identifies the virtual host. A value of zero specifies that the default virtual host should be used.

userName

A string which specifies the user name to be deleted. Usernames are not case sensitive.

Return Value

A boolean value which specifies if the virtual user has been deleted. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **DeleteUser** method removes a virtual user that was created by a previous call to the **AddUser** method. This method will not match partial usernames and wildcard characters cannot be used to delete multiple users. Usernames are not case sensitive. You cannot use this method to delete the "anonymous" user.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.DeleteUser Overload List

HttpServer.DeleteUser Method (String)

Remove a virtual user from the server.

[Visual Basic]

```
Overloads Public Function DeleteUser( _
   ByVal userName As String _
) As Boolean
```

[C#]

```
public bool DeleteUser(
   string userName
);
```

Parameters

userName

A string which specifies the user name to be deleted. Usernames are not case sensitive.

Return Value

A boolean value which specifies if the virtual user has been deleted. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **DeleteUser** method removes a virtual user that was created by a previous call to the **AddUser** method. This method will not match partial usernames and wildcard characters cannot be used to delete multiple users. Usernames are not case sensitive. You cannot use this method to delete the "anonymous" user.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.DeleteUser Overload List

HttpServer.Disconnect Method

Disconnect the specified client session from the server.

Overload List

Disconnect the specified client session from the server.

public void Disconnect();

Disconnect the specified client session from the server.

public void Disconnect(int);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.Disconnect Method (Int32)

Disconnect the specified client session from the server.

[Visual Basic]
```
Overloads Public Sub Disconnect( _
   ByVal clientId As Integer _
)
```

[C#]
```
public void Disconnect(
   int clientId
);
```

Parameters

clientId
An integer that identifies the client session.

Return Value

A boolean value which specifies if the client has been signaled to disconnect from the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Disconnect** method terminates the specified client session, releasing the socket handle other resources that were allocated for the session. It is only necessary to use this method if you want the server to explicitly terminate a client connection. Normally the client will close its connection to the server, the **OnDisconnect** event will fire and the server will automatically disconnect the client.

This method signals the thread that is managing the client that it should disconnect from the server, and it will begin the process of terminating the session. This is an asynchronous process and it is not guaranteed that the client will have actually disconnected from the server at the time that this method returns to the caller.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.Disconnect Overload List

HttpServer.Disconnect Method ()

Disconnect the specified client session from the server.

[Visual Basic]

```
Overloads Public Sub Disconnect()
```

[C#]

```
public void Disconnect();
```

Return Value

A boolean value which specifies if the client has been signaled to disconnect from the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Disconnect** method terminates the active client session, releasing the socket handle other resources that were allocated for the session. It is only necessary to use this method if you want the server to explicitly terminate a client connection. Normally the client will close its connection to the server, the **OnDisconnect** event will fire and the server will automatically disconnect the client.

This method signals the thread that is managing the client that it should disconnect from the server, and it will begin the process of terminating the session. This is an asynchronous process and it is not guaranteed that the client will have actually disconnected from the server at the time that this method returns to the caller.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.Disconnect Overload List

HttpServer.Dispose Method

Releases all resources used by HttpServer.

Overload List

Releases all resources used by HttpServer.

public void Dispose();

Releases the unmanaged resources allocated by the HttpServer class and optionally releases the managed resources.

protected void Dispose(bool);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the HttpServer class and optionally releases the managed resources.

[Visual Basic]

```
Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]

```
protected void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **HttpServer** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.Dispose Overload List

HttpServer.Dispose Method ()

Releases all resources used by HttpServer.

[Visual Basic]

```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]

```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method stops the server, terminates all active client sessions and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.Dispose Overload List

HttpServer.Finalize Method

Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.

[Visual Basic]

```
Overrides Protected Sub Finalize()
```

[C#]

```
protected override void Finalize();
```

See Also

HttpServer Class | SocketTools Namespace

HttpServer.GetAllHeaders Method

Return all of the request header values in the specified string buffer.

Overload List

Return all of the request header values in the specified string buffer.

public bool GetAllHeaders(int,ref string);

Return all of the request header values in the specified string buffer.

public bool GetAllHeaders(ref string);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.GetAllHeaders Method (Int32, String)

Return all of the request header values in the specified string buffer.

[Visual Basic]

```
Overloads Public Function GetAllHeaders( _
   ByVal clientId As Integer, _
   ByRef requestHeaders As String _
) As Boolean
```

[C#]

```
public bool GetAllHeaders(
   int clientId,
   ref string requestHeaders
);
```

Parameters

clientId

An integer that identifies the client session.

requestHeaders

A string variable that is passed by reference which will contain the request headers when the method returns.

Return Value

A boolean value which specifies if the request headers were copied to the string buffer. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **GetAllHeaders** method is used to obtain all of the request headers that were provided by the client. Each header name is separated from its value by the colon (:) and each header is terminated with a carriage return and linefeed (CRLF) sequence. Typically this method would be used within an **OnCommand** event handler. To get the value of a specific request header, use the **GetHeader** method.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.GetAllHeaders Overload List

HttpServer.GetAllHeaders Method (String)

Return all of the request header values in the specified string buffer.

```
[Visual Basic]
Overloads Public Function GetAllHeaders( _
   ByRef requestHeaders As String _
) As Boolean
```

```
[C#]
public bool GetAllHeaders(
   ref string requestHeaders
);
```

Parameters

requestHeaders
A string variable that is passed by reference which will contain the request headers when the method returns.

Return Value

A boolean value which specifies if the request headers were copied to the string buffer. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **GetAllHeaders** method is used to obtain all of the request headers that were provided by the client. Each header name is separated from its value by the colon (:) and each header is terminated with a carriage return and linefeed (CRLF) sequence. Typically this method would be used within an **OnCommand** event handler. To get the value of a specific request header, use the **GetHeader** method.

This version of the method uses the active client session and should only be called from within a server event handler. To specify a client session outside of an event handler, use the version of this method that accepts a client ID parameter.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.GetAllHeaders Overload List

HttpServer.GetHeader Method

Return the value of a request header for the specified client session.

Overload List

Return the value of a request header for the specified client session.

public bool GetHeader(int,string,ref string);

Return the value of a request header for the active client session.

public bool GetHeader(string,ref string);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.GetHeader Method (Int32, String, String)

Return the value of a request header for the specified client session.

[Visual Basic]

```
Overloads Public Function GetHeader( _
   ByVal clientId As Integer, _
   ByVal headerName As String, _
   ByRef headerValue As String _
) As Boolean
```

[C#]

```
public bool GetHeader(
   int clientId,
   string headerName,
   ref string headerValue
);
```

Parameters

clientId

An integer that identifies the client session.

headerName

A string that specifies the name of the header field. Header names are not case-sensitive and should not include the colon which acts as a delimiter that separates the header name from its value.

headerValue

A string variable that is passed by reference which will contain the value of the header when the method returns.

Return Value

A boolean value which specifies if the header value was copied to the string buffer. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **GetHeader** method will return the value of a specific header field included in the request sent by the client. Typically this is used within an **OnCommand** event handler when the server application needs to process a custom command. The **GetAllHeaders** method can be used to obtain a copy of the complete request header block submitted by the client.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.GetHeader Overload List

HttpServer.GetHeader Method (String, String)

Return the value of a request header for the active client session.

[Visual Basic]

```
Overloads Public Function GetHeader( _
   ByVal headerName As String, _
   ByRef headerValue As String _
) As Boolean
```

[C#]

```
public bool GetHeader(
   string headerName,
   ref string headerValue
);
```

Parameters

headerName

A string that specifies the name of the header field. Header names are not case-sensitive and should not include the colon which acts as a delimiter that separates the header name from its value.

headerValue

A string variable that is passed by reference which will contain the value of the header when the method returns.

Return Value

A boolean value which specifies if the header value was copied to the string buffer. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **GetHeader** method will return the value of a specific header field included in the request sent by the client. Typically this is used within an **OnCommand** event handler when the server application needs to process a custom command. The **GetAllHeaders** method can be used to obtain a copy of the complete request header block submitted by the client.

This version of the method uses the active client session and should only be called from within a server event handler. To specify a client session outside of an event handler, use the version of this method that accepts a client ID parameter.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.GetHeader Overload List

HttpServer.GetVariable Method

Return the value of a CGI environment variable for the specified client.

Overload List

Return the value of a CGI environment variable for the specified client.

public bool GetVariable(int,string,ref string);

Return the value of a CGI environment variable for the active client.

public bool GetVariable(string,ref string);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.GetVariable Method (Int32, String, String)

Return the value of a CGI environment variable for the specified client.

[Visual Basic]
```
Overloads Public Function GetVariable( _
   ByVal clientId As Integer, _
   ByVal variableName As String, _
   ByRef variableValue As String _
) As Boolean
```

[C#]
```
public bool GetVariable(
   int clientId,
   string variableName,
   ref string variableValue
);
```

Parameters

clientId
An integer that identifies the client session.

variableName
A string that that specifies the name of the environment variable. Variable names are not case-sensitive and should not include the equal sign which acts as a delimiter that separates the variable name from its value.

variableValue
A string variable that is passed by reference which will contain the value of the environment variable when the method returns.

Return Value

A boolean value which specifies if the header value was copied to the string buffer. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **GetVariable** method will return the value of an environment variable that has been defined for the client. Each client session inherits a copy of the process environment block, which is then modified to define various environment variables that are used with CGI programs and scripts. The **SetVariable** method can be used to change existing environment variables or create new variables.

The standard CGI environment variables that are defined by the server are not created until the client request has been processed. This means that environment variables such as REMOTE_ADDR and SERVER_NAME will not be defined inside an **OnConnect** event handler.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.GetVariable Overload List

HttpServer.GetVariable Method (String, String)

Return the value of a CGI environment variable for the active client.

[Visual Basic]

```
Overloads Public Function GetVariable( _
   ByVal variableName As String, _
   ByRef variableValue As String _
) As Boolean
```

[C#]

```
public bool GetVariable(
   string variableName,
   ref string variableValue
);
```

Parameters

variableName

A string that that specifies the name of the environment variable. Variable names are not case-sensitive and should not include the equal sign which acts as a delimiter that separates the variable name from its value.

variableValue

A string variable that is passed by reference which will contain the value of the environment variable when the method returns.

Return Value

A boolean value which specifies if the header value was copied to the string buffer. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **GetVariable** method will return the value of an environment variable that has been defined for the client. Each client session inherits a copy of the process environment block, which is then modified to define various environment variables that are used with CGI programs and scripts. The **SetVariable** method can be used to change existing environment variables or create new variables.

The standard CGI environment variables that are defined by the server are not created until the client request has been processed. This means that environment variables such as REMOTE_ADDR and SERVER_NAME will not be defined inside an **OnConnect** event handler.

This version of the method uses the active client session and should only be called from within a server event handler. To specify a client session outside of an event handler, use the version of this method that accepts a client ID parameter.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.GetVariable Overload List

HttpServer.Initialize Method

Initialize an instance of the HttpServer class.

Overload List

Initialize an instance of the HttpServer class.

public bool Initialize();

Initialize an instance of the HttpServer class.

public bool Initialize(string);

See Also

HttpServer Class | SocketTools Namespace | Uninitialize Method

HttpServer.Initialize Method (String)

Initialize an instance of the HttpServer class.

[Visual Basic]
```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]
```
public bool Initialize(
   string licenseKey
);
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the HttpServer class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of HttpServer can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the HttpServer class.

Example

The following example shows how to use the Initialize method to initialize an instance of the class. This example assumes that the license key string has been defined in code.

```
SocketTools.HttpServer server = new SocketTools.HttpServer();

if (server.Initialize(strLicenseKey) == false)
{
    MessageBox.Show(server.LastErrorString, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    return;
}
```

```
Dim Server As New SocketTools.HttpServer

If Server.Initialize(strLicenseKey) = False Then
    MsgBox(Server.LastErrorString, vbIconExclamation)
    Exit Sub
End If
```

See Also

HttpServer Class | SocketTools Namespace | HttpServer.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

HttpServer.Initialize Method ()

Initialize an instance of the HttpServer class.

[Visual Basic]

```
Overloads Public Function Initialize() As Boolean
```

[C#]

```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the HttpServer class, loading the networking library and allocating resources for the current thread. Typically it is not necessary to explicitly call this method because the instance of the class is initialized by the class constructor. However, if the **Uninitialize** method is called, the class must be re-initialized before any other methods are called.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.Initialize Overload List | Uninitialize Method

HttpServer.ReceiveRequest Method

Receive the request that was sent by the client to the server.

Overload List

Receive the request that was sent by the client to the server.

public bool ReceiveRequest(byte[],ref int);

Receive the request that was sent by the client to the server.

public bool ReceiveRequest(int,byte[],ref int);

Receive the request that was sent by the client to the server.

public bool ReceiveRequest(int,MemoryStream);

Receive the request that was sent by the client to the server.

public bool ReceiveRequest(int,ref string);

Receive the request that was sent by the client to the server.

public bool ReceiveRequest(MemoryStream);

Receive the request that was sent by the client to the server.

public bool ReceiveRequest(ref string);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.ReceiveRequest Method (Int32, Byte[], Int32)

Receive the request that was sent by the client to the server.

[Visual Basic]

```
Overloads Public Function ReceiveRequest( _
   ByVal clientId As Integer, _
   ByVal requestData As Byte(), _
   ByRef requestSize As Integer _
) As Boolean
```

[C#]

```
public bool ReceiveRequest(
   int clientId,
   byte[] requestData,
   ref int requestSize
);
```

Parameters

clientId
An integer that identifies the client session.

requestData
A byte array that will contain any data submitted by the client as part of the request.

requestSize
An integer value passed by reference that will contain the number of bytes copied into the byte array when the method returns.

Return Value

A boolean value which specifies if the client request was received. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **ReceiveRequest** method is called within an **OnCommand** event handler to process the command issued by the client and return information about the request to the server application. It is only necessary for the application to call this method if it wants to implement its own custom handling for a command.

It is recommended that you only use this method to process custom commands and not standard commands such as GET and POST. This ensures that the appropriate security checks are performed and the response conforms to the protocol standard. After the request data has been processed, the application should use the **SendResponse** or **SendError** method to send a response back to the client indicating success or failure.

This method may only be called once per command issued by the client.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.ReceiveRequest Overload List

HttpServer.ReceiveRequest Method (Byte[], Int32)

Receive the request that was sent by the client to the server.

[Visual Basic]
```
Overloads Public Function ReceiveRequest( _
   ByVal requestData As Byte(), _
   ByRef requestSize As Integer _
) As Boolean
```

[C#]
```
public bool ReceiveRequest(
   byte[] requestData,
   ref int requestSize
);
```

Parameters

requestData

A byte array that will contain any data submitted by the client as part of the request.

requestSize

An integer value passed by reference that will contain the number of bytes copied into the byte array when the method returns.

Return Value

A boolean value which specifies if the client request was received. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **ReceiveRequest** method is called within an **OnCommand** event handler to process the command issued by the client and return information about the request to the server application. It is only necessary for the application to call this method if it wants to implement its own custom handling for a command.

It is recommended that you only use this method to process custom commands and not standard commands such as GET and POST. This ensures that the appropriate security checks are performed and the response conforms to the protocol standard. After the request data has been processed, the application should use the **SendResponse** or **SendError** method to send a response back to the client indicating success or failure.

This method may only be called once per command issued by the client.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.ReceiveRequest Overload List

HttpServer.ReceiveRequest Method (Int32, String)

Receive the request that was sent by the client to the server.

[Visual Basic]
```
Overloads Public Function ReceiveRequest( _
   ByVal clientId As Integer, _
   ByRef requestData As String _
) As Boolean
```

[C#]
```
public bool ReceiveRequest(
   int clientId,
   ref string requestData
);
```

Parameters

clientId
An integer that identifies the client session.

requestData
A string passed by reference that will contain any data submitted by the client as part of the request.

Return Value

A boolean value which specifies if the client request was received. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **ReceiveRequest** method is called within an **OnCommand** event handler to process the command issued by the client and return information about the request to the server application. It is only necessary for the application to call this method if it wants to implement its own custom handling for a command.

It is recommended that you only use this method to process custom commands and not standard commands such as GET and POST. This ensures that the appropriate security checks are performed and the response conforms to the protocol standard. After the request data has been processed, the application should use the **SendResponse** or **SendError** method to send a response back to the client indicating success or failure.

This version of the method uses the active client session and should only be called from within a server event handler. To specify a client session outside of an event handler, use the version of this method that accepts a client ID parameter.

This method may only be called once per command issued by the client.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.ReceiveRequest Overload List

HttpServer.ReceiveRequest Method (String)

Receive the request that was sent by the client to the server.

[Visual Basic]

```
Overloads Public Function ReceiveRequest( _
   ByRef requestData As String _
) As Boolean
```

[C#]

```
public bool ReceiveRequest(
   ref string requestData
);
```

Parameters

requestData

A string passed by reference that will contain any data submitted by the client as part of the request.

Return Value

A boolean value which specifies if the client request was received. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **ReceiveRequest** method is called within an **OnCommand** event handler to process the command issued by the client and return information about the request to the server application. It is only necessary for the application to call this method if it wants to implement its own custom handling for a command.

It is recommended that you only use this method to process custom commands and not standard commands such as GET and POST. This ensures that the appropriate security checks are performed and the response conforms to the protocol standard. After the request data has been processed, the application should use the **SendResponse** or **SendError** method to send a response back to the client indicating success or failure.

This version of the method uses the active client session and should only be called from within a server event handler. To specify a client session outside of an event handler, use the version of this method that accepts a client ID parameter.

This method may only be called once per command issued by the client.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.ReceiveRequest Overload List

HttpServer.ReceiveRequest Method (Int32, MemoryStream)

Receive the request that was sent by the client to the server.

[Visual Basic]

```
Overloads Public Function ReceiveRequest( _
   ByVal clientId As Integer, _
   ByVal memStream As MemoryStream _
) As Boolean
```

[C#]

```
public bool ReceiveRequest(
   int clientId,
   MemoryStream memStream
);
```

Parameters

clientId

An integer that identifies the client session.

memStream

An instance of a MemoryStream object that will contain any data submitted by the client as part of the request.

Return Value

A boolean value which specifies if the client request was received. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **ReceiveRequest** method is called within an **OnCommand** event handler to process the command issued by the client and return information about the request to the server application. It is only necessary for the application to call this method if it wants to implement its own custom handling for a command.

It is recommended that you only use this method to process custom commands and not standard commands such as GET and POST. This ensures that the appropriate security checks are performed and the response conforms to the protocol standard. After the request data has been processed, the application should use the **SendResponse** or **SendError** method to send a response back to the client indicating success or failure.

This method may only be called once per command issued by the client.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.ReceiveRequest Overload List

HttpServer.ReceiveRequest Method (MemoryStream)

Receive the request that was sent by the client to the server.

[Visual Basic]

```
Overloads Public Function ReceiveRequest( _
   ByVal memStream As MemoryStream _
) As Boolean
```

[C#]

```
public bool ReceiveRequest(
   MemoryStream memStream
);
```

Parameters

memStream

An instance of a MemoryStream object that will contain any data submitted by the client as part of the request.

Return Value

A boolean value which specifies if the client request was received. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **ReceiveRequest** method is called within an **OnCommand** event handler to process the command issued by the client and return information about the request to the server application. It is only necessary for the application to call this method if it wants to implement its own custom handling for a command.

It is recommended that you only use this method to process custom commands and not standard commands such as GET and POST. This ensures that the appropriate security checks are performed and the response conforms to the protocol standard. After the request data has been processed, the application should use the **SendResponse** or **SendError** method to send a response back to the client indicating success or failure.

This version of the method uses the active client session and should only be called from within a server event handler. To specify a client session outside of an event handler, use the version of this method that accepts a client ID parameter.

This method may only be called once per command issued by the client.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.ReceiveRequest Overload List

HttpServer.RedirectRequest Method

Redirect the request from the client to another URL.

Overload List

Redirect the request from the client to another URL.

public bool RedirectRequest(int,string);

Redirect the request from the client to another URL.

public bool RedirectRequest(int,string,bool);

Redirect the request from the client to another URL.

public bool RedirectRequest(string);

Redirect the request from the client to another URL.

public bool RedirectRequest(string,bool);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.RedirectRequest Method (Int32, String, Boolean)

Redirect the request from the client to another URL.

[Visual Basic]

```
Overloads Public Function RedirectRequest( _
   ByVal clientId As Integer, _
   ByVal locationUrl As String, _
   ByVal isPermanent As Boolean _
) As Boolean
```

[C#]

```
public bool RedirectRequest(
   int clientId,
   string locationUrl,
   bool isPermanent
);
```

Parameters

clientId

An integer that identifies the client session.

locationUrl

A string that specifies the new location for the requested resource. This value must be a complete URL, including the http:// or https:// scheme.

isPermanent

A boolean value that specifies if the redirection should be considered temporary or permanent. If this value is true, a brower will typically cache the response and use the new resource location for subsequent requests. If this value is false, the redirection is considered to be temporary and a browser will continue to use the original resource URI.

Return Value

A boolean value which specifies if the request has been redirected. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **RedirectRequest** method can be used within an **OnCommand** event handler to redirect the client to a new location for the resource that it has requested. This redirection can be permanent or temporary, depending on whether the server expects the client to continue to use the original URL when requesting the resource.

If the ***isPermanent*** parameter is false, the actual status code that is returned to the client depends on the version of the protocol that is being used. If the client has issued the request using HTTP 1.0 then the server will return a 302 code to the client. If the client used HTTP 1.1, the server will return a 307 code to the client that indicates it should use the same command verb (GET, POST, etc.) when requesting the resource at the new location.

This method provides a simplified interface for sending a redirection status code that also implicitly sets the Location response header to the value of the ***location*** parameter. If the server application needs to send alternate redirection codes such as 305 (Use Proxy) then it should use **SetHeader** method to set the value of the Location response header, followed by the **SendReponse** method to send the redirection status code.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.RedirectRequest Overload List

HttpServer.RedirectRequest Method (Int32, String)

Redirect the request from the client to another URL.

[Visual Basic]

```
Overloads Public Function RedirectRequest( _
   ByVal clientId As Integer, _
   ByVal locationUrl As String _
) As Boolean
```

[C#]

```
public bool RedirectRequest(
   int clientId,
   string locationUrl
);
```

Parameters

clientId
An integer that identifies the client session.

locationUrl
A string that specifies the new location for the requested resource. This value must be a complete URL, including the http:// or https:// scheme.

Return Value

A boolean value which specifies if the request has been redirected. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **RedirectRequest** method can be used within an **OnCommand** event handler to redirect the client to a new location for the resource that it has requested. This redirection can be permanent or temporary, depending on whether the server expects the client to continue to use the original URL when requesting the resource.

If the ***isPermanent*** parameter is false, the actual status code that is returned to the client depends on the version of the protocol that is being used. If the client has issued the request using HTTP 1.0 then the server will return a 302 code to the client. If the client used HTTP 1.1, the server will return a 307 code to the client that indicates it should use the same command verb (GET, POST, etc.) when requesting the resource at the new location.

This method provides a simplified interface for sending a redirection status code that also implicitly sets the Location response header to the value of the ***location*** parameter. If the server application needs to send alternate redirection codes such as 305 (Use Proxy) then it should use **SetHeader** method to set the value of the Location response header, followed by the **SendReponse** method to send the redirection status code.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.RedirectRequest Overload List

HttpServer.RedirectRequest Method (String, Boolean)

Redirect the request from the client to another URL.

```
[Visual Basic]
Overloads Public Function RedirectRequest( _
   ByVal locationUrl As String, _
   ByVal isPermanent As Boolean _
) As Boolean
```

```
[C#]
public bool RedirectRequest(
   string locationUrl,
   bool isPermanent
);
```

Parameters

locationUrl
A string that specifies the new location for the requested resource. This value must be a complete URL, including the http:// or https:// scheme.

isPermanent
A boolean value that specifies if the redirection should be considered temporary or permanent. If this value is true, a brower will typically cache the response and use the new resource location for subsequent requests. If this value is false, the redirection is considered to be temporary and a browser will continue to use the original resource URI.

Return Value

A boolean value which specifies if the request has been redirected. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **RedirectRequest** method can be used within an **OnCommand** event handler to redirect the client to a new location for the resource that it has requested. This redirection can be permanent or temporary, depending on whether the server expects the client to continue to use the original URL when requesting the resource.

If the *isPermanent* parameter is false, the actual status code that is returned to the client depends on the version of the protocol that is being used. If the client has issued the request using HTTP 1.0 then the server will return a 302 code to the client. If the client used HTTP 1.1, the server will return a 307 code to the client that indicates it should use the same command verb (GET, POST, etc.) when requesting the resource at the new location.

This method provides a simplified interface for sending a redirection status code that also implicitly sets the Location response header to the value of the *location* parameter. If the server application needs to send alternate redirection codes such as 305 (Use Proxy) then it should use **SetHeader** method to set the value of the Location response header, followed by the **SendReponse** method to send the redirection status code.

This version of the method uses the active client session and should only be called from within a server event handler. To specify a client session outside of an event handler, use the version of this method that accepts a client ID parameter.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.RedirectRequest Overload List

HttpServer.RedirectRequest Method (String)

Redirect the request from the client to another URL.

```
[Visual Basic]
Overloads Public Function RedirectRequest( _
   ByVal locationUrl As String _
) As Boolean
```

```
[C#]
public bool RedirectRequest(
   string locationUrl
);
```

Parameters

locationUrl

A string that specifies the new location for the requested resource. This value must be a complete URL, including the http:// or https:// scheme.

Return Value

A boolean value which specifies if the request has been redirected. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **RedirectRequest** method can be used within an **OnCommand** event handler to redirect the client to a new location for the resource that it has requested. This redirection can be permanent or temporary, depending on whether the server expects the client to continue to use the original URL when requesting the resource.

If the ***isPermanent*** parameter is false, the actual status code that is returned to the client depends on the version of the protocol that is being used. If the client has issued the request using HTTP 1.0 then the server will return a 302 code to the client. If the client used HTTP 1.1, the server will return a 307 code to the client that indicates it should use the same command verb (GET, POST, etc.) when requesting the resource at the new location.

This method provides a simplified interface for sending a redirection status code that also implicitly sets the Location response header to the value of the ***location*** parameter. If the server application needs to send alternate redirection codes such as 305 (Use Proxy) then it should use **SetHeader** method to set the value of the Location response header, followed by the **SendReponse** method to send the redirection status code.

This version of the method uses the active client session and should only be called from within a server event handler. To specify a client session outside of an event handler, use the version of this method that accepts a client ID parameter.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.RedirectRequest Overload List

HttpServer.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. If a server has been started, it will be stopped and any active client connections will be terminated. All properties will be reset to their default values.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.Restart Method

Restarts the server and terminates all active client connections.

[Visual Basic]

```
Public Function Restart() As Boolean
```

[C#]

```
public bool Restart();
```

Return Value

A boolean value which specifies if the server was restarted. A return value of **true** specifies that the server has been successfully restarted. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Restart** method terminates all active client connections, recreates a new listening socket bound to the same address and port number, and then resumes accepting new client connections.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.RegisterHandler Method

Register a CGI program for use and associate it with a file name extension.

Overload List

Register a CGI program for use and associate it with a file name extension.

public bool RegisterHandler(int,string,string);

Register a CGI program for use and associate it with a file name extension.

public bool RegisterHandler(int,string,string,string);

Register a CGI program for use and associate it with a file name extension.

public bool RegisterHandler(int,string,string,string,string);

Register a CGI program for use and associate it with a file name extension.

public bool RegisterHandler(string,string);

Register a CGI program for use and associate it with a file name extension.

public bool RegisterHandler(string,string,string);

Register a CGI program for use and associate it with a file name extension.

public bool RegisterHandler(string,string,string,string);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.RegisterHandler Method (Int32, String, String, String, String)

Register a CGI program for use and associate it with a file name extension.

[Visual Basic]

```
Overloads Public Function RegisterHandler( _
   ByVal hostId As Integer, _
   ByVal extension As String, _
   ByVal program As String, _
   ByVal parameters As String, _
   ByVal directory As String _
) As Boolean
```

[C#]

```
public bool RegisterHandler(
   int hostId,
   string extension,
   string program,
   string parameters,
   string directory
);
```

Parameters

hostId
An integer value which identifies the virtual host. A value of zero specifies that the default virtual host should be used.

extension
A string which specifies the file name extension that is associated with the CGI program.

program
A string which specifies the full path to the CGI program on the local system.

parameters
A string that specifies additional parameters for the program. This value will be passed to the program as command line arguments. If the CGI program does not require any command line parameters, this parameter may be an empty string.

directory
A string that specifies the current working directory for the program. If this parameter is an empty string, the program will use the root directory of the virtual host as the current working directory.

Return Value

A boolean value which specifies if the CGI handler was registered. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **RegisterHandler** method registers an executable CGI program and associates it with a file name extension. When the client issues a GET or POST command that specifies a file with that extension, the program will be executed and the output return to the client.

The ***program*** string specifies file name of the CGI program. You should not install any executable programs in the server root directory or its subdirectories. A client should never have the ability to directly access the executable file itself. It is permitted to have multiple file name extensions that reference the same program. The only requirement is that the extension be unique for the given host. The program name may contain environment variables surrounded by % symbols. For example, %ProgramFiles% would be expanded to the C:\Program Files folder.

It is important to note that the program specified by ***ProgramFile*** must be an executable file, not a script or batch file. If the program name does not contain a directory path, then the standard Windows pathing rules will be used when searching for an executable file that matches the given name. It is recommended that you always provide a full path to the executable file.

The ***parameters*** string can specify additional command line parameters that should be passed to the CGI program as arguments. This string can also contain a placeholder named "%1" that will be replaced by the full path to the local script filename. If no parameters are specified, the script file name will be passed to the program as its only argument.

The executable program that is registered using this program must be a console application that conforms to the CGI/1.1 specification defined in RFC 3875. Request data submitted by the client as part of a POST will be provided to the program as standard input. The output from the program must be written to standard output. The first lines of output from the program should be any response headers, followed by an empty line. Each line should be terminated with a carriage-return and linefeed (CRLF) sequence. If the CGI program outputs additional data to be processed by the client, it should provide Content-Type and Content-Length response headers.

When developing a CGI program, it is important to take into consideration the environment that it will be executing in. The program will be started as a child process of the server application, and will inherit the same privileges. This means that it will typically have access to the boot drive, the Windows folders and the system registry. CGI programs must ensure that all query parameters and request data submitted by the client have been validated.

If the server is running on a system with User Account Control (UAC) enabled and does not have elevated privileges, do not register a program that requires elevated privileges or has a manifest that specifies the requestedExecutionLevel as requiring administrative privileges.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.RegisterHandler Overload List

HttpServer.RegisterHandler Method (Int32, String, String, String)

Register a CGI program for use and associate it with a file name extension.

[Visual Basic]

```
Overloads Public Function RegisterHandler( _
   ByVal hostId As Integer, _
   ByVal extension As String, _
   ByVal program As String, _
   ByVal parameters As String _
) As Boolean
```

[C#]

```
public bool RegisterHandler(
   int hostId,
   string extension,
   string program,
   string parameters
);
```

Parameters

hostId
: An integer value which identifies the virtual host. A value of zero specifies that the default virtual host should be used.

extension
: A string which specifies the file name extension that is associated with the CGI program.

program
: A string which specifies the full path to the CGI program on the local system.

parameters
: A string that specifies additional parameters for the program. This value will be passed to the program as command line arguments. If the CGI program does not require any command line parameters, this parameter may be an empty string.

Return Value

A boolean value which specifies if the CGI handler was registered. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **RegisterHandler** method registers an executable CGI program and associates it with a file name extension. When the client issues a GET or POST command that specifies a file with that extension, the program will be executed and the output return to the client.

The ***program*** string specifies file name of the CGI program. You should not install any executable programs in the server root directory or its subdirectories. A client should never have the ability to directly access the executable file itself. It is permitted to have multiple file name extensions that reference the same program. The only requirement is that the extension be unique for the given host. The program name may contain environment variables surrounded by % symbols. For example, %ProgramFiles% would be expanded to the C:\Program Files folder.

It is important to note that the program specified by ***ProgramFile*** must be an executable file, not a script or batch file. If the program name does not contain a directory path, then the standard Windows pathing rules will be used when searching for an executable file that matches the given name. It is recommended that you always provide a full path to the executable file.

The ***parameters*** string can specify additional command line parameters that should be passed to the CGI program as arguments. This string can also contain a placeholder named "%1" that will be replaced by the full path to the local script filename. If no parameters are specified, the script file name will be passed to the program as its only argument.

The executable program that is registered using this program must be a console application that conforms to the CGI/1.1 specification defined in RFC 3875. Request data submitted by the client as part of a POST will be provided to the program as standard input. The output from the program must be written to standard output. The first lines of output from the program should be any response headers, followed by an empty line. Each line should be terminated with a carriage-return and linefeed (CRLF) sequence. If the CGI program outputs additional data to be processed by the client, it should provide Content-Type and Content-Length response headers.

When developing a CGI program, it is important to take into consideration the environment that it will be executing in. The program will be started as a child process of the server application, and will inherit the same privileges. This means that it will typically have access to the boot drive, the Windows folders and the system registry. CGI programs must ensure that all query parameters and request data submitted by the client have been validated.

If the server is running on a system with User Account Control (UAC) enabled and does not have elevated privileges, do not register a program that requires elevated privileges or has a manifest that specifies the requestedExecutionLevel as requiring administrative privileges.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.RegisterHandler Overload List

HttpServer.RegisterHandler Method (Int32, String, String)

Register a CGI program for use and associate it with a file name extension.

[Visual Basic]

```
Overloads Public Function RegisterHandler( _
   ByVal hostId As Integer, _
   ByVal extension As String, _
   ByVal program As String _
) As Boolean
```

[C#]

```
public bool RegisterHandler(
   int hostId,
   string extension,
   string program
);
```

Parameters

hostId
An integer value which identifies the virtual host. A value of zero specifies that the default virtual host should be used.

extension
A string which specifies the file name extension that is associated with the CGI program.

program
A string which specifies the full path to the CGI program on the local system.

Return Value

A boolean value which specifies if the CGI handler was registered. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **RegisterHandler** method registers an executable CGI program and associates it with a file name extension. When the client issues a GET or POST command that specifies a file with that extension, the program will be executed and the output return to the client.

The ***program*** string specifies file name of the CGI program. You should not install any executable programs in the server root directory or its subdirectories. A client should never have the ability to directly access the executable file itself. It is permitted to have multiple file name extensions that reference the same program. The only requirement is that the extension be unique for the given host. The program name may contain environment variables surrounded by % symbols. For example, %ProgramFiles% would be expanded to the C:\Program Files folder.

It is important to note that the program specified by ***ProgramFile*** must be an executable file, not a script or batch file. If the program name does not contain a directory path, then the standard Windows pathing rules will be used when searching for an executable file that matches the given name. It is recommended that you always provide a full path to the executable file.

The ***parameters*** string can specify additional command line parameters that should be passed to the CGI program as arguments. This string can also contain a placeholder named "%1" that will be replaced by the full path to the local script filename. If no parameters are specified, the script file name will be passed to the program as its only argument.

The executable program that is registered using this program must be a console application that conforms to the CGI/1.1 specification defined in RFC 3875. Request data submitted by the client as part of a POST will be provided to the program as standard input. The output from the program must be written to standard output. The first lines of output from the program should be any response headers, followed by an empty line. Each line should be terminated with a carriage-return and linefeed (CRLF) sequence. If the CGI program outputs additional data to be processed by the client, it should provide Content-Type and Content-Length response headers.

When developing a CGI program, it is important to take into consideration the environment that it will be executing in. The program will be started as a child process of the server application, and will inherit the same privileges. This means that it will typically have access to the boot drive, the Windows folders and the system registry. CGI programs must ensure that all query parameters and request data submitted by the client have been validated.

If the server is running on a system with User Account Control (UAC) enabled and does not have elevated privileges, do not register a program that requires elevated privileges or has a manifest that specifies the requestedExecutionLevel as requiring administrative privileges.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.RegisterHandler Overload List

HttpServer.RegisterHandler Method (String, String, String, String)

Register a CGI program for use and associate it with a file name extension.

[Visual Basic]

```
Overloads Public Function RegisterHandler( _
   ByVal extension As String, _
   ByVal program As String, _
   ByVal parameters As String, _
   ByVal directory As String _
) As Boolean
```

[C#]

```
public bool RegisterHandler(
   string extension,
   string program,
   string parameters,
   string directory
);
```

Parameters

extension
A string which specifies the file name extension that is associated with the CGI program.

program
A string which specifies the full path to the CGI program on the local system.

parameters
A string that specifies additional parameters for the program. This value will be passed to the program as command line arguments. If the CGI program does not require any command line parameters, this parameter may be an empty string.

directory
A string that specifies the current working directory for the program. If this parameter is an empty string, the program will use the root directory of the virtual host as the current working directory.

Return Value

A boolean value which specifies if the CGI handler was registered. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **RegisterHandler** method registers an executable CGI program and associates it with a file name extension. When the client issues a GET or POST command that specifies a file with that extension, the program will be executed and the output return to the client.

The ***program*** string specifies file name of the CGI program. You should not install any executable programs in the server root directory or its subdirectories. A client should never have the ability to directly access the executable file itself. It is permitted to have multiple file name extensions that reference the same program. The only requirement is that the extension be unique for the given host. The program name may contain environment variables surrounded by % symbols. For example, %ProgramFiles% would be expanded to the C:\Program Files folder.

It is important to note that the program specified by ***ProgramFile*** must be an executable file, not a script or batch file. If the program name does not contain a directory path, then the standard Windows pathing rules will be used when searching for an executable file that matches the given name. It is recommended that you always provide a full path to the executable file.

The ***parameters*** string can specify additional command line parameters that should be passed to the CGI program as arguments. This string can also contain a placeholder named "%1" that will be replaced by the full path to the local script filename. If no parameters are specified, the script file name will be passed to the program as its only argument.

The executable program that is registered using this program must be a console application that conforms to the CGI/1.1 specification defined in RFC 3875. Request data submitted by the client as part of a POST will be provided to the program as standard input. The output from the program must be written to standard output. The first lines of output from the program should be any response headers, followed by an empty line. Each line should be terminated with a carriage-return and linefeed (CRLF) sequence. If the CGI program outputs additional data to be processed by the client, it should provide Content-Type and Content-Length response headers.

When developing a CGI program, it is important to take into consideration the environment that it will be executing in. The program will be started as a child process of the server application, and will inherit the same privileges. This means that it will typically have access to the boot drive, the Windows folders and the system registry. CGI programs must ensure that all query parameters and request data submitted by the client have been validated.

If the server is running on a system with User Account Control (UAC) enabled and does not have elevated privileges, do not register a program that requires elevated privileges or has a manifest that specifies the requestedExecutionLevel as requiring administrative privileges.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.RegisterHandler Overload List

HttpServer.RegisterHandler Method (String, String, String)

Register a CGI program for use and associate it with a file name extension.

```
[Visual Basic]
Overloads Public Function RegisterHandler( _
   ByVal extension As String, _
   ByVal program As String, _
   ByVal parameters As String _
) As Boolean
```

```
[C#]
public bool RegisterHandler(
   string extension,
   string program,
   string parameters
);
```

Parameters

extension
A string which specifies the file name extension that is associated with the CGI program.

program
A string which specifies the full path to the CGI program on the local system.

parameters
A string that specifies additional parameters for the program. This value will be passed to the program as command line arguments. If the CGI program does not require any command line parameters, this parameter may be an empty string.

Return Value

A boolean value which specifies if the CGI handler was registered. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **RegisterHandler** method registers an executable CGI program and associates it with a file name extension. When the client issues a GET or POST command that specifies a file with that extension, the program will be executed and the output return to the client.

The ***program*** string specifies file name of the CGI program. You should not install any executable programs in the server root directory or its subdirectories. A client should never have the ability to directly access the executable file itself. It is permitted to have multiple file name extensions that reference the same program. The only requirement is that the extension be unique for the given host. The program name may contain environment variables surrounded by % symbols. For example, %ProgramFiles% would be expanded to the C:\Program Files folder.

It is important to note that the program specified by ***ProgramFile*** must be an executable file, not a script or batch file. If the program name does not contain a directory path, then the standard Windows pathing rules will be used when searching for an executable file that matches the given name. It is recommended that you always provide a full path to the executable file.

The ***parameters*** string can specify additional command line parameters that should be passed to the CGI program as arguments. This string can also contain a placeholder named "%1" that will be replaced by the full path to the local script filename. If no parameters are specified, the script file name will be passed to the program as its only argument.

The executable program that is registered using this program must be a console application that conforms to the CGI/1.1 specification defined in RFC 3875. Request data submitted by the client as part of a POST will be provided to the program as standard input. The output from the program must be written to standard output. The first lines of output from the program should be any response headers, followed by an empty line. Each line should be terminated with a carriage-return and linefeed (CRLF) sequence. If the CGI program outputs additional data to be processed by the client, it should provide Content-Type and Content-Length response headers.

When developing a CGI program, it is important to take into consideration the environment that it will be executing in. The program will be started as a child process of the server application, and will inherit the same privileges. This means that it will typically have access to the boot drive, the Windows folders and the system registry. CGI programs must ensure that all query parameters and request data submitted by the client have been validated.

If the server is running on a system with User Account Control (UAC) enabled and does not have elevated privileges, do not register a program that requires elevated privileges or has a manifest that specifies the requestedExecutionLevel as requiring administrative privileges.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.RegisterHandler Overload List

HttpServer.RegisterHandler Method (String, String)

Register a CGI program for use and associate it with a file name extension.

[Visual Basic]

```
Overloads Public Function RegisterHandler( _
   ByVal extension As String, _
   ByVal program As String _
) As Boolean
```

[C#]

```
public bool RegisterHandler(
   string extension,
   string program
);
```

Parameters

extension
A string which specifies the file name extension that is associated with the CGI program.

program
A string which specifies the full path to the CGI program on the local system.

Return Value

A boolean value which specifies if the CGI handler was registered. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **RegisterHandler** method registers an executable CGI program and associates it with a file name extension. When the client issues a GET or POST command that specifies a file with that extension, the program will be executed and the output return to the client.

The ***program*** string specifies file name of the CGI program. You should not install any executable programs in the server root directory or its subdirectories. A client should never have the ability to directly access the executable file itself. It is permitted to have multiple file name extensions that reference the same program. The only requirement is that the extension be unique for the given host. The program name may contain environment variables surrounded by % symbols. For example, %ProgramFiles% would be expanded to the C:\Program Files folder.

It is important to note that the program specified by ***ProgramFile*** must be an executable file, not a script or batch file. If the program name does not contain a directory path, then the standard Windows pathing rules will be used when searching for an executable file that matches the given name. It is recommended that you always provide a full path to the executable file.

The ***parameters*** string can specify additional command line parameters that should be passed to the CGI program as arguments. This string can also contain a placeholder named "%1" that will be replaced by the full path to the local script filename. If no parameters are specified, the script file name will be passed to the program as its only argument.

The executable program that is registered using this program must be a console application that conforms to the CGI/1.1 specification defined in RFC 3875. Request data submitted by the client as part of a POST will be provided to the program as standard input. The output from the program must be written to standard output. The first lines of output from the program should be any response headers, followed by an empty line. Each line should be terminated with a carriage-return and linefeed (CRLF) sequence. If the

CGI program outputs additional data to be processed by the client, it should provide Content-Type and Content-Length response headers.

When developing a CGI program, it is important to take into consideration the environment that it will be executing in. The program will be started as a child process of the server application, and will inherit the same privileges. This means that it will typically have access to the boot drive, the Windows folders and the system registry. CGI programs must ensure that all query parameters and request data submitted by the client have been validated.

If the server is running on a system with User Account Control (UAC) enabled and does not have elevated privileges, do not register a program that requires elevated privileges or has a manifest that specifies the requestedExecutionLevel as requiring administrative privileges.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.RegisterHandler Overload List

HttpServer.RegisterProgram Method

Register a CGI program for use and associate it with a virtual path on the server.

Overload List

Register a CGI program for use and associate it with a virtual path on the server.

public bool RegisterProgram(int,string,string);

Register a CGI program for use and associate it with a virtual path on the server.

public bool RegisterProgram(int,string,string,string);

Register a CGI program for use and associate it with a virtual path on the server.

public bool RegisterProgram(int,string,string,string,string);

Register a CGI program for use and associate it with a virtual path on the server.

public bool RegisterProgram(string,string);

Register a CGI program for use and associate it with a virtual path on the server.

public bool RegisterProgram(string,string,string);

Register a CGI program for use and associate it with a virtual path on the server.

public bool RegisterProgram(string,string,string,string);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.RegisterProgram Method (Int32, String, String, String, String)

Register a CGI program for use and associate it with a virtual path on the server.

[Visual Basic]

```
Overloads Public Function RegisterProgram( _
   ByVal hostId As Integer, _
   ByVal command As String, _
   ByVal program As String, _
   ByVal parameters As String, _
   ByVal directory As String _
) As Boolean
```

[C#]

```
public bool RegisterProgram(
   int hostId,
   string command,
   string program,
   string parameters,
   string directory
);
```

Parameters

hostId
An integer value which identifies the virtual host. A value of zero specifies that the default virtual host should be used.

command
A string which specifies the virtual path to the CGI program. This must be an absolute path, but does not have to specify a pre-existing virtual path or map to the directory structure of the root document directory for the server. The maximum length of the virtual path is 1024 characters.

program
A string which specifies the full path to the CGI program on the local system.

parameters
A string that specifies additional parameters for the program. This value will be passed to the program as command line arguments. If the CGI program does not require any command line parameters, this parameter may be an empty string.

directory
An optional string that specifies the current working directory for the program. If this parameter is an empty string, the server will use the root document directory for the virtual host.

Return Value

A boolean value which specifies if the CGI program was registered with the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **RegisterProgram** method registers a CGI program and associates it with a virtual path. When the client issues a GET or POST command specifying the virtual path associated with the program, the program will be executed and the output return to the client.

The ***program*** string specifies file name of the CGI program. You should not install any executable programs in the server root directory or its subdirectories. A client should never have the ability to directly access the executable file itself. It is permitted to have multiple virtual paths that reference the same executable file. The only requirement is that the virtual path be unique for the given host. The program name may contain environment variables surrounded by % symbols. For example, %ProgramFiles% would be expanded to the C:\Program Files folder.

It is important to note that the program specified by ***program*** must be an executable file, not a script or batch file. If the program name does not contain a directory path, then the standard Windows pathing rules will be used when searching for an executable file that matches the given name. It is recommended that you always provide a full path to the executable file.

The ***parameters*** string can specify additional command line parameters that should be passed to the CGI program as arguments. This string can also contain a placeholder named "%1" that will be replaced by the virtual path associated with the program. If this argument is omitted, no additional parameters are passed to the program.

The executable program that is registered using this program must be a console application that conforms to the CGI/1.1 specification defined in RFC 3875. Request data submitted by the client as part of a POST will be provided to the program as standard input. The output from the program must be written to standard output. The first lines of output from the program should be any response headers, followed by an empty line. Each line should be terminated with a carriage-return and linefeed (CRLF) sequence. If the CGI program outputs additional data to be processed by the client, it should provide Content-Type and Content-Length response headers.

When developing a CGI program, it is important to take into consideration the environment that it will be executing in. The program will be started as a child process of the server application, and will inherit the same privileges. This means that it will typically have access to the boot drive, the Windows folders and the system registry. CGI programs must ensure that all query parameters and request data submitted by the client have been validated.

If the server is running on a system with User Account Control (UAC) enabled and does not have elevated privileges, do not register a program that requires elevated privileges or has a manifest that specifies the requestedExecutionLevel as requiring administrative privileges.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.RegisterProgram Overload List

HttpServer.RegisterProgram Method (Int32, String, String, String)

Register a CGI program for use and associate it with a virtual path on the server.

[Visual Basic]

```
Overloads Public Function RegisterProgram( _
   ByVal hostId As Integer, _
   ByVal command As String, _
   ByVal program As String, _
   ByVal parameters As String _
) As Boolean
```

[C#]

```
public bool RegisterProgram(
   int hostId,
   string command,
   string program,
   string parameters
);
```

Parameters

hostId

An integer value which identifies the virtual host. A value of zero specifies that the default virtual host should be used.

command

A string which specifies the virtual path to the CGI program. This must be an absolute path, but does not have to specify a pre-existing virtual path or map to the directory structure of the root document directory for the server. The maximum length of the virtual path is 1024 characters.

program

A string which specifies the full path to the CGI program on the local system.

parameters

A string that specifies additional parameters for the program. This value will be passed to the program as command line arguments. If the CGI program does not require any command line parameters, this parameter may be an empty string.

Return Value

A boolean value which specifies if the CGI program was registered with the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **RegisterProgram** method registers a CGI program and associates it with a virtual path. When the client issues a GET or POST command specifying the virtual path associated with the program, the program will be executed and the output return to the client.

The ***program*** string specifies file name of the CGI program. You should not install any executable programs in the server root directory or its subdirectories. A client should never have the ability to directly access the executable file itself. It is permitted to have multiple virtual paths that reference the same executable file. The only requirement is that the virtual path be unique for the given host. The program name may contain environment variables surrounded by % symbols. For example, %ProgramFiles% would be expanded to the C:\Program Files folder.

It is important to note that the program specified by ***program*** must be an executable file, not a script or batch file. If the program name does not contain a directory path, then the standard Windows pathing rules will be used when searching for an executable file that matches the given name. It is recommended that you always provide a full path to the executable file.

The ***parameters*** string can specify additional command line parameters that should be passed to the CGI program as arguments. This string can also contain a placeholder named "%1" that will be replaced by the virtual path associated with the program. If this argument is omitted, no additional parameters are passed to the program.

The executable program that is registered using this program must be a console application that conforms to the CGI/1.1 specification defined in RFC 3875. Request data submitted by the client as part of a POST will be provided to the program as standard input. The output from the program must be written to standard output. The first lines of output from the program should be any response headers, followed by an empty line. Each line should be terminated with a carriage-return and linefeed (CRLF) sequence. If the CGI program outputs additional data to be processed by the client, it should provide Content-Type and Content-Length response headers.

When developing a CGI program, it is important to take into consideration the environment that it will be executing in. The program will be started as a child process of the server application, and will inherit the same privileges. This means that it will typically have access to the boot drive, the Windows folders and the system registry. CGI programs must ensure that all query parameters and request data submitted by the client have been validated.

If the server is running on a system with User Account Control (UAC) enabled and does not have elevated privileges, do not register a program that requires elevated privileges or has a manifest that specifies the requestedExecutionLevel as requiring administrative privileges.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.RegisterProgram Overload List

HttpServer.RegisterProgram Method (Int32, String, String)

Register a CGI program for use and associate it with a virtual path on the server.

[Visual Basic]

```
Overloads Public Function RegisterProgram( _
   ByVal hostId As Integer, _
   ByVal command As String, _
   ByVal program As String _
) As Boolean
```

[C#]

```
public bool RegisterProgram(
   int hostId,
   string command,
   string program
);
```

Parameters

hostId

An integer value which identifies the virtual host. A value of zero specifies that the default virtual host should be used.

command

A string which specifies the virtual path to the CGI program. This must be an absolute path, but does not have to specify a pre-existing virtual path or map to the directory structure of the root document directory for the server. The maximum length of the virtual path is 1024 characters.

program

A string which specifies the full path to the CGI program on the local system.

Return Value

A boolean value which specifies if the CGI program was registered with the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **RegisterProgram** method registers a CGI program and associates it with a virtual path. When the client issues a GET or POST command specifying the virtual path associated with the program, the program will be executed and the output return to the client.

The ***program*** string specifies file name of the CGI program. You should not install any executable programs in the server root directory or its subdirectories. A client should never have the ability to directly access the executable file itself. It is permitted to have multiple virtual paths that reference the same executable file. The only requirement is that the virtual path be unique for the given host. The program name may contain environment variables surrounded by % symbols. For example, %ProgramFiles% would be expanded to the C:\Program Files folder.

It is important to note that the program specified by ***program*** must be an executable file, not a script or batch file. If the program name does not contain a directory path, then the standard Windows pathing rules will be used when searching for an executable file that matches the given name. It is recommended that you always provide a full path to the executable file.

The ***parameters*** string can specify additional command line parameters that should be passed to the CGI program as arguments. This string can also contain a placeholder named "%1" that will be replaced by the virtual path associated with the program. If this argument is omitted, no additional parameters are passed to the program.

The executable program that is registered using this program must be a console application that conforms to the CGI/1.1 specification defined in RFC 3875. Request data submitted by the client as part of a POST will be provided to the program as standard input. The output from the program must be written to standard output. The first lines of output from the program should be any response headers, followed by an empty line. Each line should be terminated with a carriage-return and linefeed (CRLF) sequence. If the CGI program outputs additional data to be processed by the client, it should provide Content-Type and Content-Length response headers.

When developing a CGI program, it is important to take into consideration the environment that it will be executing in. The program will be started as a child process of the server application, and will inherit the same privileges. This means that it will typically have access to the boot drive, the Windows folders and the system registry. CGI programs must ensure that all query parameters and request data submitted by the client have been validated.

If the server is running on a system with User Account Control (UAC) enabled and does not have elevated privileges, do not register a program that requires elevated privileges or has a manifest that specifies the requestedExecutionLevel as requiring administrative privileges.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.RegisterProgram Overload List

HttpServer.RegisterProgram Method (String, String, String, String)

Register a CGI program for use and associate it with a virtual path on the server.

[Visual Basic]

```
Overloads Public Function RegisterProgram( _
   ByVal command As String, _
   ByVal program As String, _
   ByVal parameters As String, _
   ByVal directory As String _
) As Boolean
```

[C#]

```
public bool RegisterProgram(
   string command,
   string program,
   string parameters,
   string directory
);
```

Parameters

command

A string which specifies the virtual path to the CGI program. This must be an absolute path, but does not have to specify a pre-existing virtual path or map to the directory structure of the root document directory for the server. The maximum length of the virtual path is 1024 characters.

program

A string which specifies the full path to the CGI program on the local system.

parameters

A string that specifies additional parameters for the program. This value will be passed to the program as command line arguments. If the CGI program does not require any command line parameters, this parameter may be an empty string.

directory

An optional string that specifies the current working directory for the program. If this parameter is an empty string, the server will use the root document directory for the virtual host.

Return Value

A boolean value which specifies if the CGI program was registered with the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **RegisterProgram** method registers a CGI program and associates it with a virtual path. When the client issues a GET or POST command specifying the virtual path associated with the program, the program will be executed and the output return to the client.

The ***program*** string specifies file name of the CGI program. You should not install any executable programs in the server root directory or its subdirectories. A client should never have the ability to directly access the executable file itself. It is permitted to have multiple virtual paths that reference the same executable file. The only requirement is that the virtual path be unique for the given host. The program name may contain environment variables surrounded by % symbols. For example, %ProgramFiles% would be expanded to the C:\Program Files folder.

It is important to note that the program specified by ***program*** must be an executable file, not a script or batch file. If the program name does not contain a directory path, then the standard Windows pathing rules will be used when searching for an executable file that matches the given name. It is recommended that you always provide a full path to the executable file.

The ***parameters*** string can specify additional command line parameters that should be passed to the CGI program as arguments. This string can also contain a placeholder named "%1" that will be replaced by the virtual path associated with the program. If this argument is omitted, no additional parameters are passed to the program.

The executable program that is registered using this program must be a console application that conforms to the CGI/1.1 specification defined in RFC 3875. Request data submitted by the client as part of a POST will be provided to the program as standard input. The output from the program must be written to standard output. The first lines of output from the program should be any response headers, followed by an empty line. Each line should be terminated with a carriage-return and linefeed (CRLF) sequence. If the CGI program outputs additional data to be processed by the client, it should provide Content-Type and Content-Length response headers.

When developing a CGI program, it is important to take into consideration the environment that it will be executing in. The program will be started as a child process of the server application, and will inherit the same privileges. This means that it will typically have access to the boot drive, the Windows folders and the system registry. CGI programs must ensure that all query parameters and request data submitted by the client have been validated.

If the server is running on a system with User Account Control (UAC) enabled and does not have elevated privileges, do not register a program that requires elevated privileges or has a manifest that specifies the requestedExecutionLevel as requiring administrative privileges.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.RegisterProgram Overload List

HttpServer.RegisterProgram Method (String, String, String)

Register a CGI program for use and associate it with a virtual path on the server.

[Visual Basic]

```
Overloads Public Function RegisterProgram( _
   ByVal command As String, _
   ByVal program As String, _
   ByVal parameters As String _
) As Boolean
```

[C#]

```
public bool RegisterProgram(
   string command,
   string program,
   string parameters
);
```

Parameters

command

A string which specifies the virtual path to the CGI program. This must be an absolute path, but does not have to specify a pre-existing virtual path or map to the directory structure of the root document directory for the server. The maximum length of the virtual path is 1024 characters.

program

A string which specifies the full path to the CGI program on the local system.

parameters

A string that specifies additional parameters for the program. This value will be passed to the program as command line arguments. If the CGI program does not require any command line parameters, this parameter may be an empty string.

Return Value

A boolean value which specifies if the CGI program was registered with the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **RegisterProgram** method registers a CGI program and associates it with a virtual path. When the client issues a GET or POST command specifying the virtual path associated with the program, the program will be executed and the output return to the client.

The ***program*** string specifies file name of the CGI program. You should not install any executable programs in the server root directory or its subdirectories. A client should never have the ability to directly access the executable file itself. It is permitted to have multiple virtual paths that reference the same executable file. The only requirement is that the virtual path be unique for the given host. The program name may contain environment variables surrounded by % symbols. For example, %ProgramFiles% would be expanded to the C:\Program Files folder.

It is important to note that the program specified by ***program*** must be an executable file, not a script or batch file. If the program name does not contain a directory path, then the standard Windows pathing rules will be used when searching for an executable file that matches the given name. It is recommended that you always provide a full path to the executable file.

The ***parameters*** string can specify additional command line parameters that should be passed to the CGI program as arguments. This string can also contain a placeholder named "%1" that will be replaced by the virtual path associated with the program. If this argument is omitted, no additional parameters are passed to the program.

The executable program that is registered using this program must be a console application that conforms to the CGI/1.1 specification defined in RFC 3875. Request data submitted by the client as part of a POST will be provided to the program as standard input. The output from the program must be written to standard output. The first lines of output from the program should be any response headers, followed by an empty line. Each line should be terminated with a carriage-return and linefeed (CRLF) sequence. If the CGI program outputs additional data to be processed by the client, it should provide Content-Type and Content-Length response headers.

When developing a CGI program, it is important to take into consideration the environment that it will be executing in. The program will be started as a child process of the server application, and will inherit the same privileges. This means that it will typically have access to the boot drive, the Windows folders and the system registry. CGI programs must ensure that all query parameters and request data submitted by the client have been validated.

If the server is running on a system with User Account Control (UAC) enabled and does not have elevated privileges, do not register a program that requires elevated privileges or has a manifest that specifies the requestedExecutionLevel as requiring administrative privileges.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.RegisterProgram Overload List

HttpServer.RegisterProgram Method (String, String)

Register a CGI program for use and associate it with a virtual path on the server.

[Visual Basic]

```
Overloads Public Function RegisterProgram( _
   ByVal command As String, _
   ByVal program As String _
) As Boolean
```

[C#]

```
public bool RegisterProgram(
   string command,
   string program
);
```

Parameters

command

A string which specifies the virtual path to the CGI program. This must be an absolute path, but does not have to specify a pre-existing virtual path or map to the directory structure of the root document directory for the server. The maximum length of the virtual path is 1024 characters.

program

A string which specifies the full path to the CGI program on the local system.

Return Value

A boolean value which specifies if the CGI program was registered with the server. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **RegisterProgram** method registers a CGI program and associates it with a virtual path. When the client issues a GET or POST command specifying the virtual path associated with the program, the program will be executed and the output return to the client.

The ***program*** string specifies file name of the CGI program. You should not install any executable programs in the server root directory or its subdirectories. A client should never have the ability to directly access the executable file itself. It is permitted to have multiple virtual paths that reference the same executable file. The only requirement is that the virtual path be unique for the given host. The program name may contain environment variables surrounded by % symbols. For example, %ProgramFiles% would be expanded to the C:\Program Files folder.

It is important to note that the program specified by ***program*** must be an executable file, not a script or batch file. If the program name does not contain a directory path, then the standard Windows pathing rules will be used when searching for an executable file that matches the given name. It is recommended that you always provide a full path to the executable file.

The ***parameters*** string can specify additional command line parameters that should be passed to the CGI program as arguments. This string can also contain a placeholder named "%1" that will be replaced by the virtual path associated with the program. If this argument is omitted, no additional parameters are passed to the program.

The executable program that is registered using this program must be a console application that conforms to the CGI/1.1 specification defined in RFC 3875. Request data submitted by the client as part of a POST will be provided to the program as standard input. The output from the program must be written to standard output. The first lines of output from the program should be any response headers, followed by an empty line. Each line should be terminated with a carriage-return and linefeed (CRLF) sequence. If the CGI program outputs additional data to be processed by the client, it should provide Content-Type and Content-Length response headers.

When developing a CGI program, it is important to take into consideration the environment that it will be executing in. The program will be started as a child process of the server application, and will inherit the same privileges. This means that it will typically have access to the boot drive, the Windows folders and the system registry. CGI programs must ensure that all query parameters and request data submitted by the client have been validated.

If the server is running on a system with User Account Control (UAC) enabled and does not have elevated privileges, do not register a program that requires elevated privileges or has a manifest that specifies the requestedExecutionLevel as requiring administrative privileges.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.RegisterProgram Overload List

HttpServer.RequireAuthentication Method

Send a response to the client indicating that authentication is required.

Overload List

Send a response to the client indicating that authentication is required.

public bool RequireAuthentication();

Send a response to the client indicating that authentication is required.

public bool RequireAuthentication(int);

Send a response to the client indicating that authentication is required.

public bool RequireAuthentication(int,string);

Send a response to the client indicating that authentication is required.

public bool RequireAuthentication(string);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.RequireAuthentication Method (Int32, String)

Send a response to the client indicating that authentication is required.

[Visual Basic]

```
Overloads Public Function RequireAuthentication( _
   ByVal clientId As Integer, _
   ByVal realm As String _
) As Boolean
```

[C#]

```
public bool RequireAuthentication(
   int clientId,
   string realm
);
```

Parameters

clientId

An integer that identifies the client session.

realm

A string value that is displayed a web browser to indicate to the user which username and password they should use. If this parameter is omitted or is an empty string, the domain name the client used to establish the connection will be used.

Return Value

A boolean value which specifies if the response was sent to the client. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **RequireAuthentication** method can be used within an **OnCommand** event handler to indicate to the client that it must provide a username and password to access the requested resource. The client should respond by issuing another request that includes the required credentials. To determine if a client has included valid credentials with its request, check the value of the **IsAuthenticated** property.

Some clients may require that the session be secure if authentication is requested or display warning messages to the user if the connection is not secure. If your application will require clients to authenticate before accessing specific resources, it is recommended that you enable security by setting the **Secure** property to **True** prior to starting the server.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.RequireAuthentication Overload List

HttpServer.RequireAuthentication Method (Int32)

Send a response to the client indicating that authentication is required.

[Visual Basic]
```
Overloads Public Function RequireAuthentication( _
   ByVal clientId As Integer _
) As Boolean
```

[C#]
```
public bool RequireAuthentication(
   int clientId
);
```

Parameters

clientId
An integer that identifies the client session.

Return Value

A boolean value which specifies if the response was sent to the client. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **RequireAuthentication** method can be used within an **OnCommand** event handler to indicate to the client that it must provide a username and password to access the requested resource. The client should respond by issuing another request that includes the required credentials. To determine if a client has included valid credentials with its request, check the value of the **IsAuthenticated** property.

Some clients may require that the session be secure if authentication is requested or display warning messages to the user if the connection is not secure. If your application will require clients to authenticate before accessing specific resources, it is recommended that you enable security by setting the **Secure** property to **True** prior to starting the server.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.RequireAuthentication Overload List

HttpServer.RequireAuthentication Method (String)

Send a response to the client indicating that authentication is required.

```
[Visual Basic]
Overloads Public Function RequireAuthentication( _
   ByVal realm As String _
) As Boolean
```

```
[C#]
public bool RequireAuthentication(
   string realm
);
```

Parameters

realm

A string value that is displayed a web browser to indicate to the user which username and password they should use. If this parameter is omitted or is an empty string, the domain name the client used to establish the connection will be used.

Return Value

A boolean value which specifies if the response was sent to the client. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **RequireAuthentication** method can be used within an **OnCommand** event handler to indicate to the client that it must provide a username and password to access the requested resource. The client should respond by issuing another request that includes the required credentials. To determine if a client has included valid credentials with its request, check the value of the **IsAuthenticated** property.

Some clients may require that the session be secure if authentication is requested or display warning messages to the user if the connection is not secure. If your application will require clients to authenticate before accessing specific resources, it is recommended that you enable security by setting the **Secure** property to **True** prior to starting the server.

This version of the method uses the active client session and should only be called from within a server event handler. To specify a client session outside of an event handler, use the version of this method that accepts a client ID parameter.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.RequireAuthentication Overload List

HttpServer.RequireAuthentication Method ()

Send a response to the client indicating that authentication is required.

[Visual Basic]

```
Overloads Public Function RequireAuthentication() As Boolean
```

[C#]

```
public bool RequireAuthentication();
```

Return Value

A boolean value which specifies if the response was sent to the client. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **RequireAuthentication** method can be used within an **OnCommand** event handler to indicate to the client that it must provide a username and password to access the requested resource. The client should respond by issuing another request that includes the required credentials. To determine if a client has included valid credentials with its request, check the value of the **IsAuthenticated** property.

Some clients may require that the session be secure if authentication is requested or display warning messages to the user if the connection is not secure. If your application will require clients to authenticate before accessing specific resources, it is recommended that you enable security by setting the **Secure** property to **True** prior to starting the server.

This version of the method uses the active client session and should only be called from within a server event handler. To specify a client session outside of an event handler, use the version of this method that accepts a client ID parameter.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.RequireAuthentication Overload List

HttpServer.ResolvePath Method

Resolve a path to its full virtual or local file name.

Overload List

Resolve a path to its full virtual or local file name.

public bool ResolvePath(int,string,ref string,bool);

Resolve a path to its full virtual or local file name.

public bool ResolvePath(string,ref string,bool);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.ResolvePath Method (Int32, String, String, Boolean)

Resolve a path to its full virtual or local file name.

[Visual Basic]

```
Overloads Public Function ResolvePath( _
   ByVal clientId As Integer, _
   ByVal sourcePath As String, _
   ByRef resolvedPath As String, _
   ByVal isVirtual As Boolean _
) As Boolean
```

[C#]

```
public bool ResolvePath(
   int clientId,
   string sourcePath,
   ref string resolvedPath,
   bool isVirtual
);
```

Parameters

clientId
An integer that identifies the client session.

sourcePath
A string that specifies the name of the path to resolve. This may either be a virtual path, or a path to a local file name or directory.

resolvedPath
A string that will contain the resolved path when the method returns.

isVirtual
A Boolean parameter that specifies if the source path is a virtual path or local path.

Return Value

A boolean value which specifies if the source path could be resolved. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **ResolvePath** method is used to resolve a local file name or directory to obtain its virtual path name, or obtain the full path name of a file or directory that is mapped to a virtual path. If the ***isVirtual*** parameter is **false**, the ***sourcePath*** parameter is considered to be a path to a local file or directory and the ***resolvedPath*** parameter will contain the virtual path. If the ***isVirtual*** parameter is **true**, then the s***ourcePath*** parameter is considered to be a virtual path and the ***resolvedPath*** parameter will contain the full path to the local file or directory that the virtual path is mapped to

A virtual path for the client is either relative to the server root directory, or the client home directory if the client was authenticated as a restricted user. These virtual paths are what the client will see as an absolute path on the server. For example, if the server was configured to use "C:\ProgramData\MyServer" as the root directory, and the ***SourcePath*** parameter was specified as "C:\ProgramData\MyServer\Documents\Research", this method would return the virtual path to that directory as "/Documents/Research".

If the client session was authenticated as a restricted user, then the virtual path is always relative to the client home directory instead of the server root directory. This is because restricted users are isolated to their own home directory and any subdirectories. For example, if restricted user "John" has a home directory of "C:\ProgramData\MyServer\Users\John" and the ***SourcePath*** parameter was specified as "C:\ProgramData\MyServer\Users\John\Accounting\Projections.pdf" this method would return the virtual path as "/Accounting/Projections.pdf".

If the ***sourcePath*** parameter specifies a file or directory outside of the server root directory, this method will fail and the last error code will be set to **errorInvalidFileName**. This method can only be used with authenticated clients. If the ***clientId*** parameter specifies a client session that has not been authenticated, this method will fail and the last error code will be **errorAuthenticationRequired**.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.ResolvePath Overload List

HttpServer.ResolvePath Method (String, String, Boolean)

Resolve a path to its full virtual or local file name.

[Visual Basic]

```
Overloads Public Function ResolvePath( _
   ByVal sourcePath As String, _
   ByRef resolvedPath As String, _
   ByVal isVirtual As Boolean _
) As Boolean
```

[C#]

```
public bool ResolvePath(
   string sourcePath,
   ref string resolvedPath,
   bool isVirtual
);
```

Parameters

sourcePath
: A string that specifies the name of the path to resolve. This may either be a virtual path, or a path to a local file name or directory.

resolvedPath
: A string that will contain the resolved path when the method returns.

isVirtual
: A Boolean parameter that specifies if the source path is a virtual path or local path.

Return Value

A boolean value which specifies if the source path could be resolved. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **ResolvePath** method is used to resolve a local file name or directory to obtain its virtual path name, or obtain the full path name of a file or directory that is mapped to a virtual path. If the ***isVirtual*** parameter is **false**, the ***sourcePath*** parameter is considered to be a path to a local file or directory and the ***resolvedPath*** parameter will contain the virtual path. If the ***isVirtual*** parameter is **true**, then the s***ourcePath*** parameter is considered to be a virtual path and the ***resolvedPath*** parameter will contain the full path to the local file or directory that the virtual path is mapped to

A virtual path for the client is either relative to the server root directory, or the client home directory if the client was authenticated as a restricted user. These virtual paths are what the client will see as an absolute path on the server. For example, if the server was configured to use "C:\ProgramData\MyServer" as the root directory, and the ***SourcePath*** parameter was specified as "C:\ProgramData\MyServer\Documents\Research", this method would return the virtual path to that directory as "/Documents/Research".

If the client session was authenticated as a restricted user, then the virtual path is always relative to the client home directory instead of the server root directory. This is because restricted users are isolated to their own home directory and any subdirectories. For example, if restricted user "John" has a home directory of "C:\ProgramData\MyServer\Users\John" and the ***SourcePath*** parameter was specified as "C:\ProgramData\MyServer\Users\John\Accounting\Projections.pdf" this method would return the virtual path as "/Accounting/Projections.pdf".

If the ***sourcePath*** parameter specifies a file or directory outside of the server root directory, this method will fail and the last error code will be set to **errorInvalidFileName**. This method can only be used with authenticated clients. If the ***clientId*** parameter specifies a client session that has not been authenticated, this method will fail and the last error code will be **errorAuthenticationRequired**.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.ResolvePath Overload List

HttpServer.Resume Method

Resume accepting new client connections.

[Visual Basic]

```
Public Function Resume() As Boolean
```

[C#]

```
public bool Resume();
```

Return Value

A boolean value which specifies if the server has resumed accepting client connections. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Resume** method instructs the server to resume accepting new client connections. Any pending client connections that were requested while the server was suspended will be accepted.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.SendError Method

Send an error result code and message to the client in response to a command.

Overload List

Send an error result code and message to the client in response to a command.

public bool SendError(int);

Send an error result code and message to the client in response to a command.

public bool SendError(int,int);

Send an error result code and message to the client in response to a command.

public bool SendError(int,int,string);

Send an error result code and message to the client in response to a command.

public bool SendError(int,string);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.SendError Method (Int32, Int32, String)

Send an error result code and message to the client in response to a command.

[Visual Basic]

```
Overloads Public Function SendError( _
   ByVal clientId As Integer, _
   ByVal resultCode As Integer, _
   ByVal message As String _
) As Boolean
```

[C#]

```
public bool SendError(
   int clientId,
   int resultCode,
   string message
);
```

Parameters

clientId

An integer that identifies the client session.

resultCode

An integer value that specifies the error code that should be sent to the client. This value should correspond to the error result codes defined for HTTP in RFC 2616, which are three-digit values in the range of 400 through 599. The method will fail if an invalid error code is specified.

message

An string value that specifies a message to be sent to the client. If this parameter is an empty string, a default message associated with the result code will be used.

Return Value

A boolean value which specifies if the error response was sent to the client. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **SendError** method sends a response to the client indicating that an error has occurred, providing a numeric error code and HTML formatted text which may be displayed to the user. The ***Message*** parameter should provide a brief description of the error that will be included in the output sent to the client. Note that the message should not contain any special formatting control characters or HTML markup.

This method provides a simplified interface for sending an error response to the client. In some cases, a browser may choose to display its own error message to the user in place of the generic HTML document generated by this method. If you want your application to send a customized HTML document for a specific type of error, you should use the **SendResponse** method.

If you wish to redirect the client to use an alternate URL to access the requested resource, it is recommended that you use the **RedirectRequest** method rather than sending an error response.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.SendError Overload List

HttpServer.SendError Method (Int32, Int32)

Send an error result code and message to the client in response to a command.

[Visual Basic]
```
Overloads Public Function SendError( _
   ByVal clientId As Integer, _
   ByVal resultCode As Integer _
) As Boolean
```

[C#]
```
public bool SendError(
   int clientId,
   int resultCode
);
```

Parameters

clientId

An integer that identifies the client session.

resultCode

An integer value that specifies the error code that should be sent to the client. This value should correspond to the error result codes defined for HTTP in RFC 2616, which are three-digit values in the range of 400 through 599. The method will fail if an invalid error code is specified.

Return Value

A boolean value which specifies if the error response was sent to the client. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **SendError** method sends a response to the client indicating that an error has occurred, providing a numeric error code and HTML formatted text which may be displayed to the user. The ***Message*** parameter should provide a brief description of the error that will be included in the output sent to the client. Note that the message should not contain any special formatting control characters or HTML markup.

This method provides a simplified interface for sending an error response to the client. In some cases, a browser may choose to display its own error message to the user in place of the generic HTML document generated by this method. If you want your application to send a customized HTML document for a specific type of error, you should use the **SendResponse** method.

If you wish to redirect the client to use an alternate URL to access the requested resource, it is recommended that you use the **RedirectRequest** method rather than sending an error response.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.SendError Overload List

HttpServer.SendError Method (Int32, String)

Send an error result code and message to the client in response to a command.

[Visual Basic]

```
Overloads Public Function SendError( _
   ByVal resultCode As Integer, _
   ByVal message As String _
) As Boolean
```

[C#]

```
public bool SendError(
   int resultCode,
   string message
);
```

Parameters

resultCode

An integer value that specifies the error code that should be sent to the client. This value should correspond to the error result codes defined for HTTP in RFC 2616, which are three-digit values in the range of 400 through 599. The method will fail if an invalid error code is specified.

message

An string value that specifies a message to be sent to the client. If this parameter is an empty string, a default message associated with the result code will be used.

Return Value

A boolean value which specifies if the error response was sent to the client. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **SendError** method sends a response to the client indicating that an error has occurred, providing a numeric error code and HTML formatted text which may be displayed to the user. The ***Message*** parameter should provide a brief description of the error that will be included in the output sent to the client. Note that the message should not contain any special formatting control characters or HTML markup.

This method provides a simplified interface for sending an error response to the client. In some cases, a browser may choose to display its own error message to the user in place of the generic HTML document generated by this method. If you want your application to send a customized HTML document for a specific type of error, you should use the **SendResponse** method.

If you wish to redirect the client to use an alternate URL to access the requested resource, it is recommended that you use the **RedirectRequest** method rather than sending an error response.

This version of the method uses the active client session and should only be called from within a server event handler. To specify a client session outside of an event handler, use the version of this method that accepts a client ID parameter.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.SendError Overload List

HttpServer.SendError Method (Int32)

Send an error result code and message to the client in response to a command.

[Visual Basic]

```
Overloads Public Function SendError( _
   ByVal resultCode As Integer _
) As Boolean
```

[C#]

```
public bool SendError(
   int resultCode
);
```

Parameters

resultCode

An integer value that specifies the error code that should be sent to the client. This value should correspond to the error result codes defined for HTTP in RFC 2616, which are three-digit values in the range of 400 through 599. The method will fail if an invalid error code is specified.

Return Value

A boolean value which specifies if the error response was sent to the client. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **SendError** method sends a response to the client indicating that an error has occurred, providing a numeric error code and HTML formatted text which may be displayed to the user. The ***Message*** parameter should provide a brief description of the error that will be included in the output sent to the client. Note that the message should not contain any special formatting control characters or HTML markup.

This method provides a simplified interface for sending an error response to the client. In some cases, a browser may choose to display its own error message to the user in place of the generic HTML document generated by this method. If you want your application to send a customized HTML document for a specific type of error, you should use the **SendResponse** method.

If you wish to redirect the client to use an alternate URL to access the requested resource, it is recommended that you use the **RedirectRequest** method rather than sending an error response.

This version of the method uses the active client session and should only be called from within a server event handler. To specify a client session outside of an event handler, use the version of this method that accepts a client ID parameter.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.SendError Overload List

HttpServer.SendResponse Method

Send a result code to the client in response to a command.

Overload List

Send a result code to the client in response to a command.

public bool SendResponse(int);

Send a result code and message to the client in response to a command.

public bool SendResponse(int,byte[],int);

Send a result code to the client in response to a command.

public bool SendResponse(int,int);

Send a result code and message to the client in response to a command.

public bool SendResponse(int,int,byte[],int);

Send a result code and message to the client in response to a command.

public bool SendResponse(int,int,MemoryStream);

Send a result code and message to the client in response to a command.

public bool SendResponse(int,int,string);

Send a result code and message to the client in response to a command.

public bool SendResponse(int,MemoryStream);

Send a result code and message to the client in response to a command.

public bool SendResponse(int,string);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.SendResponse Method (Int32, Int32, Byte[], Int32)

Send a result code and message to the client in response to a command.

[Visual Basic]

```
Overloads Public Function SendResponse( _
   ByVal clientId As Integer, _
   ByVal resultCode As Integer, _
   ByVal responseData As Byte(), _
   ByVal responseSize As Integer _
) As Boolean
```

[C#]

```
public bool SendResponse(
   int clientId,
   int resultCode,
   byte[] responseData,
   int responseSize
);
```

Parameters

clientId
An integer that identifies the client session.

resultCode
An integer value that specifies the command result code to be returned to the client.

responseData
A byte array that contains data that should be returned to the client in response to a request.

responseSize
An integer value that specifies the number of bytes of data that should be sent to the client.

Return Value

A boolean value which specifies if the response was sent to the client. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **SendResponse** method is used to respond to a command issued by the client from within an **OnCommand** event handler. Command responses are normally handled by the server as a normal part of processing a command and this method is only used if the application has implemented custom commands or wishes to modify the standard responses sent by the server.

Result codes must be three digits (in the range of 100 through 999) and although this method will support the use of non-standard result codes, it is recommended that the client application use the standard codes defined in RFC 2616 whenever possible. The use of non-standard result codes may cause problems with HTTP clients that expect specific result codes in response to a particular command.

If you do not wish to return any data to the client in response to its request (for example, if you want the response to only consist of the headers set using the **SetHeader** method), then you can use the overloaded version of this method that omits the ***responseData*** parameter, and you should specify a result code of 204. This tells the client that the request was successful and there is no data included with the response.

This method should only be called once in response to a command sent by the client. If a result code has already been sent in response to a command and this method is called, it will fail and return an error. This is necessary because sending multiple result codes in response to a single command may cause unpredictable behavior by the client.

This version of the method would typically be used when returning binary data to the client. For textual data, such as an HTML or XML response, it is recommended that you use the overloaded version of this method that accepts a string.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.SendResponse Overload List

HttpServer.SendResponse Method (Int32, Byte[], Int32)

Send a result code and message to the client in response to a command.

[Visual Basic]

```
Overloads Public Function SendResponse( _
   ByVal resultCode As Integer, _
   ByVal responseData As Byte(), _
   ByVal responseSize As Integer _
) As Boolean
```

[C#]

```
public bool SendResponse(
   int resultCode,
   byte[] responseData,
   int responseSize
);
```

Parameters

resultCode
An integer value that specifies the command result code to be returned to the client.

responseData
A byte array that contains data that should be returned to the client in response to a request.

responseSize
An integer value that specifies the number of bytes of data that should be sent to the client.

Return Value

A boolean value which specifies if the response was sent to the client. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **SendResponse** method is used to respond to a command issued by the client from within an **OnCommand** event handler. Command responses are normally handled by the server as a normal part of processing a command and this method is only used if the application has implemented custom commands or wishes to modify the standard responses sent by the server.

Result codes must be three digits (in the range of 100 through 999) and although this method will support the use of non-standard result codes, it is recommended that the client application use the standard codes defined in RFC 2616 whenever possible. The use of non-standard result codes may cause problems with HTTP clients that expect specific result codes in response to a particular command.

If you do not wish to return any data to the client in response to its request (for example, if you want the response to only consist of the headers set using the **SetHeader** method), then you can use the overloaded version of this method that omits the ***responseData*** parameter, and you should specify a result code of 204. This tells the client that the request was successful and there is no data included with the response.

This method should only be called once in response to a command sent by the client. If a result code has already been sent in response to a command and this method is called, it will fail and return an error. This is necessary because sending multiple result codes in response to a single command may cause unpredictable behavior by the client.

This version of the method would typically be used when returning binary data to the client. For textual data, such as an HTML or XML response, it is recommended that you use the overloaded version of this method that accepts a string.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.SendResponse Overload List

HttpServer.SendResponse Method (Int32, Int32, String)

Send a result code and message to the client in response to a command.

[Visual Basic]

```
Overloads Public Function SendResponse( _
   ByVal clientId As Integer, _
   ByVal resultCode As Integer, _
   ByVal responseData As String _
) As Boolean
```

[C#]

```
public bool SendResponse(
   int clientId,
   int resultCode,
   string responseData
);
```

Parameters

clientId
An integer that identifies the client session.

resultCode
An integer value that specifies the command result code to be returned to the client.

responseData
A string that contains data that should be returned to the client in response to a request.

Return Value

A boolean value which specifies if the response was sent to the client. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **SendResponse** method is used to respond to a command issued by the client from within an **OnCommand** event handler. Command responses are normally handled by the server as a normal part of processing a command and this method is only used if the application has implemented custom commands or wishes to modify the standard responses sent by the server.

Result codes must be three digits (in the range of 100 through 999) and although this method will support the use of non-standard result codes, it is recommended that the client application use the standard codes defined in RFC 2616 whenever possible. The use of non-standard result codes may cause problems with HTTP clients that expect specific result codes in response to a particular command.

If you do not wish to return any data to the client in response to its request (for example, if you want the response to only consist of the headers set using the **SetHeader** method), then you can use the overloaded version of this method that omits the ***responseData*** parameter, and you should specify a result code of 204. This tells the client that the request was successful and there is no data included with the response.

This method should only be called once in response to a command sent by the client. If a result code has already been sent in response to a command and this method is called, it will fail and return an error. This is necessary because sending multiple result codes in response to a single command may cause unpredictable behavior by the client.

This version of the method would typically be used when returning textual data to the client. For binary data, it is recommended that you use the overloaded version of this method that accepts a byte array.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.SendResponse Overload List

HttpServer.SendResponse Method (Int32, String)

Send a result code and message to the client in response to a command.

[Visual Basic]

```
Overloads Public Function SendResponse( _
   ByVal resultCode As Integer, _
   ByVal responseData As String _
) As Boolean
```

[C#]

```
public bool SendResponse(
   int resultCode,
   string responseData
);
```

Parameters

resultCode

An integer value that specifies the command result code to be returned to the client.

responseData

A string that contains data that should be returned to the client in response to a request.

Return Value

A boolean value which specifies if the response was sent to the client. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **SendResponse** method is used to respond to a command issued by the client from within an **OnCommand** event handler. Command responses are normally handled by the server as a normal part of processing a command and this method is only used if the application has implemented custom commands or wishes to modify the standard responses sent by the server.

Result codes must be three digits (in the range of 100 through 999) and although this method will support the use of non-standard result codes, it is recommended that the client application use the standard codes defined in RFC 2616 whenever possible. The use of non-standard result codes may cause problems with HTTP clients that expect specific result codes in response to a particular command.

If you do not wish to return any data to the client in response to its request (for example, if you want the response to only consist of the headers set using the **SetHeader** method), then you can use the overloaded version of this method that omits the ***responseData*** parameter, and you should specify a result code of 204. This tells the client that the request was successful and there is no data included with the response.

This method should only be called once in response to a command sent by the client. If a result code has already been sent in response to a command and this method is called, it will fail and return an error. This is necessary because sending multiple result codes in response to a single command may cause unpredictable behavior by the client.

This version of the method would typically be used when returning textual data to the client. For binary data, it is recommended that you use the overloaded version of this method that accepts a byte array.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.SendResponse Overload List

HttpServer.SendResponse Method (Int32, Int32, MemoryStream)

Send a result code and message to the client in response to a command.

[Visual Basic]

```
Overloads Public Function SendResponse( _
   ByVal clientId As Integer, _
   ByVal resultCode As Integer, _
   ByVal memStream As MemoryStream _
) As Boolean
```

[C#]

```
public bool SendResponse(
   int clientId,
   int resultCode,
   MemoryStream memStream
);
```

Parameters

clientId

An integer that identifies the client session.

resultCode

An integer value that specifies the command result code to be returned to the client.

memStream

A MemoryStream object that contains data that should be returned to the client in response to a request.

Return Value

A boolean value which specifies if the response was sent to the client. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **SendResponse** method is used to respond to a command issued by the client from within an **OnCommand** event handler. Command responses are normally handled by the server as a normal part of processing a command and this method is only used if the application has implemented custom commands or wishes to modify the standard responses sent by the server.

Result codes must be three digits (in the range of 100 through 999) and although this method will support the use of non-standard result codes, it is recommended that the client application use the standard codes defined in RFC 2616 whenever possible. The use of non-standard result codes may cause problems with HTTP clients that expect specific result codes in response to a particular command.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.SendResponse Overload List

HttpServer.SendResponse Method (Int32, MemoryStream)

Send a result code and message to the client in response to a command.

[Visual Basic]

```
Overloads Public Function SendResponse( _
   ByVal resultCode As Integer, _
   ByVal memStream As MemoryStream _
) As Boolean
```

[C#]

```
public bool SendResponse(
   int resultCode,
   MemoryStream memStream
);
```

Parameters

resultCode

An integer value that specifies the command result code to be returned to the client.

memStream

A MemoryStream object that contains data that should be returned to the client in response to a request.

Return Value

A boolean value which specifies if the response was sent to the client. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **SendResponse** method is used to respond to a command issued by the client from within an **OnCommand** event handler. Command responses are normally handled by the server as a normal part of processing a command and this method is only used if the application has implemented custom commands or wishes to modify the standard responses sent by the server.

Result codes must be three digits (in the range of 100 through 999) and although this method will support the use of non-standard result codes, it is recommended that the client application use the standard codes defined in RFC 2616 whenever possible. The use of non-standard result codes may cause problems with HTTP clients that expect specific result codes in response to a particular command.

If you do not wish to return any data to the client in response to its request (for example, if you want the response to only consist of the headers set using the **SetHeader** method), then you can use the overloaded version of this method that omits the ***buffer*** and ***length*** parameters, and should specify a result code of 204. This tells the client that the request was successful and there is no data included with the response.

This method should only be called once in response to a command sent by the client. If a result code has already been sent in response to a command and this method is called, it will fail and return an error. This is necessary because sending multiple result codes in response to a single command may cause unpredictable behavior by the client.

This version of the method would typically be used when returning binary data to the client. For textual data, such as an HTML or XML response, it is recommended that you use the overloaded version of this method that accepts a string.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.SendResponse Overload List

HttpServer.SendResponse Method (Int32, Int32)

Send a result code to the client in response to a command.

[Visual Basic]

```
Overloads Public Function SendResponse( _
   ByVal clientId As Integer, _
   ByVal resultCode As Integer _
) As Boolean
```

[C#]

```
public bool SendResponse(
   int clientId,
   int resultCode
);
```

Parameters

clientId
An integer that identifies the client session.

resultCode
An integer value that specifies the command result code to be returned to the client.

Return Value

A boolean value which specifies if the response was sent to the client. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **SendResponse** method is used to respond to a command issued by the client from within an **OnCommand** event handler. Command responses are normally handled by the server as a normal part of processing a command and this method is only used if the application has implemented custom commands or wishes to modify the standard responses sent by the server.

Result codes must be three digits (in the range of 100 through 999) and although this method will support the use of non-standard result codes, it is recommended that the client application use the standard codes defined in RFC 2616 whenever possible. The use of non-standard result codes may cause problems with HTTP clients that expect specific result codes in response to a particular command.

This version of the method would be used when you do not need to provide any response data to the client. For successful responses, it is recommended that you specify a result code of 204 which tells the client that the response will not include any data.

To send an error response to the client, it is recommended that you use the **SendError** method. This will ensure that a valid error response will be sent to the client, including a default HTML page that can be displayed by a browser to notify a user of the error condition.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.SendResponse Overload List

HttpServer.SetHeader Method

Create or change the value of a response header for the client session.

Overload List

Create or change the value of a response header for the client session.

public bool SetHeader(int,string,string);

Create or change the value of a response header for the client session.

public bool SetHeader(string,string);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.SetHeader Method (Int32, String, String)

Create or change the value of a response header for the client session.

[Visual Basic]

```
Overloads Public Function SetHeader( _
   ByVal clientId As Integer, _
   ByVal headerName As String, _
   ByVal headerValue As String _
) As Boolean
```

[C#]

```
public bool SetHeader(
   int clientId,
   string headerName,
   string headerValue
);
```

Parameters

clientId
An integer that identifies the client session.

headerName
A string that specifies the name of the header field. Header names are not case-sensitive and should not include the colon which acts as a delimiter that separates the header name from its value.

headerValue
A string variable that contains the new value of the response header.

Return Value

A boolean value which specifies if the response header was created or modified. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **SetHeader** method will change the value of a response header for the specified client session, typically within an **OnCommand** event handler. If the ***headerName*** value matches an existing header field, its value will be replaced. If the header name is not defined, then a new header will be created with the given value. You should not change the value of most standard response header values unless you are certain of the impact that it would have on the normal operation of the client.

If you wish to define a custom header value that would be included in the response to a client request, you should prefix the header name with "X-" to avoid potential conflicts with the standard response headers. For example, if you wanted to identify a customer, you could create a header field with the name "X-Customer-ID" and set the value to the customer ID number. The client application would receive this custom header value as part of the response to its request for a document.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.SetHeader Overload List

HttpServer.SetHeader Method (String, String)

Create or change the value of a response header for the client session.

[Visual Basic]

```
Overloads Public Function SetHeader( _
   ByVal headerName As String, _
   ByVal headerValue As String _
) As Boolean
```

[C#]

```
public bool SetHeader(
   string headerName,
   string headerValue
);
```

Parameters

headerName

A string that specifies the name of the header field. Header names are not case-sensitive and should not include the colon which acts as a delimiter that separates the header name from its value.

headerValue

A string variable that contains the new value of the response header.

Return Value

A boolean value which specifies if the response header was created or modified. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **SetHeader** method will change the value of a response header for the specified client session, typically within an **OnCommand** event handler. If the ***headerName*** value matches an existing header field, its value will be replaced. If the header name is not defined, then a new header will be created with the given value. You should not change the value of most standard response header values unless you are certain of the impact that it would have on the normal operation of the client.

If you wish to define a custom header value that would be included in the response to a client request, you should prefix the header name with "X-" to avoid potential conflicts with the standard response headers. For example, if you wanted to identify a customer, you could create a header field with the name "X-Customer-ID" and set the value to the customer ID number. The client application would receive this custom header value as part of the response to its request for a document.

This version of the method uses the active client session and should only be called from within a server event handler. To specify a client session outside of an event handler, use the version of this method that accepts a client ID parameter.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.SetHeader Overload List

HttpServer.SetVariable Method

Create or change the value of a CGI environment variable for the specified client.

Overload List

Create or change the value of a CGI environment variable for the specified client.

public bool SetVariable(int,string,string);

Create or change the value of a CGI environment variable for the specified client.

public bool SetVariable(string,string);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.SetVariable Method (Int32, String, String)

Create or change the value of a CGI environment variable for the specified client.

```
[Visual Basic]
Overloads Public Function SetVariable( _
   ByVal clientId As Integer, _
   ByVal variableName As String, _
   ByVal variableValue As String _
) As Boolean
```

```
[C#]
public bool SetVariable(
   int clientId,
   string variableName,
   string variableValue
);
```

Parameters

clientId
An integer that identifies the client session.

variableName
A string that specifies the name of the header field. Header names are not case-sensitive and should not include the colon which acts as a delimiter that separates the header name from its value.

variableValue
A string variable that contains the new value of the response header.

Return Value

A boolean value which specifies if the environment variable was created or modified. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **SetVariable** method will change the value of a environment variable for the specified client session, typically within an **OnCommand** event handler. If the ***variableName*** value matches an existing variable, its value will be replaced. If the variable is not defined, then a new variable will be created with the given value. The value of an environment variable can be obtained using the **GetVariable** function.

The server will automatically create a number of different environment variables that will be passed to a program or script executed by the server. These variables are defined in RFC 3875 as part of the Common Gateway Interface (CGI) 1.1 specification. The following variables are defined by the server and should not be modified directly by the application:

AUTH_TYPE	The authorization scheme used by the server to authenticate the client session.
CONTENT_LENGTH	The length of the request data provided by the client.
CONTENT_TYPE	The MIME type that identifies the type of content provided by the client.
DOCUMENT_ROOT	The full path to the local document root directory on the server.
GATEWAY_INTERFACE	The version of the Common Gateway Interface that is being used by the server.

PATH_INFO	The resource or sub-resource that is to be returned by the program or script.
PATH_TRANSLATED	The path information mapped to the server root document directory structure.
QUERY_STRING	The URL encoded query parameters passed to the program or script.
REMOTE_ADDR	The network address of the client sending the request to the server.
REMOTE_HOST	The same value as the REMOTE_ADDR variable.
REMOTE_USER	The username specified as part of the authentication credentials provided by the client.
REQUEST_METHOD	The method used by the client to request the resource.
REQUEST_URI	The URI for the script provided by the client.
SCRIPT_FILENAME	The full path to the program or script on the server.
SCRIPT_NAME	The path to the program or script specified by the client.
SERVER_NAME	The hostname or IP address of the server that the client connected to.
SERVER_PORT	The port number that the client used to connect to the server.
SERVER_PORT_SECURE	This variable has a value of "1" if the client connection to the server is secure.
SERVER_PROTOCOL	The version of the server protocol used.
SERVER_SOFTWARE	The server identity string which specifies the application name and version.

In addition to the environment variables listed, the server will also create variables that are prefixed with "HTTP_" that are set to the value of request headers that are not otherwise defined. For example, the HTTP_USER_AGENT variable will be set to the value of the User-Agent header provided by the client as part of the request.

Note that calling the **SetVariable** method from within the **OnExecute** event handler will have no effect because it occurs after the CGI program or script has completed execution. To create or modify environment variables for the client session, it should be done within an **OnCommand** event handler.

This method will not change the environment block for the server process. Each client session is allocated its own private environment block which is inherited by the CGI program. When the client session terminates, the memory allocated for its environment is released.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.SetVariable Overload List

HttpServer.SetVariable Method (String, String)

Create or change the value of a CGI environment variable for the specified client.

[Visual Basic]

```
Overloads Public Function SetVariable( _
   ByVal variableName As String, _
   ByVal variableValue As String _
) As Boolean
```

[C#]

```
public bool SetVariable(
   string variableName,
   string variableValue
);
```

Parameters

variableName

A string that specifies the name of the header field. Header names are not case-sensitive and should not include the colon which acts as a delimiter that separates the header name from its value.

variableValue

A string variable that contains the new value of the response header.

Return Value

A boolean value which specifies if the environment variable was created or modified. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **SetVariable** method will change the value of a environment variable for the specified client session, typically within an **OnCommand** event handler. If the ***variableName*** value matches an existing variable, its value will be replaced. If the variable is not defined, then a new variable will be created with the given value. The value of an environment variable can be obtained using the **GetVariable** method.

Note that calling the **SetVariable** method from within the **OnExecute** event handler will have no effect because it occurs after the CGI program or script has completed execution. To create or modify environment variables for the client session, it should be done within an **OnCommand** event handler.

This method will not change the environment block for the server process. Each client session is allocated its own private environment block which is inherited by the CGI program. When the client session terminates, the memory allocated for its environment is released.

This version of the method uses the active client session and should only be called from within a server event handler. To specify a client session outside of an event handler, use the version of this method that accepts a client ID parameter.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.SetVariable Overload List

HttpServer.Start Method

Start listening for client connections on the specified IP address and port number.

Overload List

Start listening for client connections on the specified IP address and port number.

public bool Start();

Start listening for client connections on the specified IP address and port number.

public bool Start(int);

Start listening for client connections on the specified IP address and port number.

public bool Start(string,int);

Start listening for client connections on the specified IP address and port number.

public bool Start(string,int,string);

Start listening for client connections on the specified IP address and port number.

public bool Start(string,int,string,int);

Start listening for client connections on the specified IP address and port number.

public bool Start(string,int,string,int,int);

Start listening for client connections on the specified IP address and port number.

public bool Start(string,int,string,int,int,ServerOptions);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.Start Method (String, Int32, String, Int32, Int32, ServerOptions)

Start listening for client connections on the specified IP address and port number.

[Visual Basic]
```
Overloads Public Function Start( _
   ByVal localAddress As String, _
   ByVal localPort As Integer, _
   ByVal rootDirectory As String, _
   ByVal maxClients As Integer, _
   ByVal idleTime As Integer, _
   ByVal options As ServerOptions _
) As Boolean
```

[C#]
```
public bool Start(
   string localAddress,
   int localPort,
   string rootDirectory,
   int maxClients,
   int idleTime,
   ServerOptions options
);
```

Parameters

localAddress

A string which specifies the local hostname or IP address address that the server should be bound to. If this parameter is an empty string, then an appropriate address will automatically be used. If a specific address is used, the server will only accept client connections on the network interface that is bound to that address.

localPort

An integer that specifies the port number the server should use to listen for client connections. If a value of zero is specified, the server will use the standard port number 80 to listen for connections, or port 443 if the server is configured to use implicit SSL. The port number used by the application must be unique and multiple instances of a server cannot use the same port number. It is recommended that a port number greater than 5000 be used for private, application-specific implementations.

rootDirectory

A string that specifies the path to the root directory for the server. If this value is an empty string, the server will use the current working directory as the root directory.

maxClients

An integer value that specifies the maximum number of clients that may connect to the server.

idleTime

An integer value that specifies the number of seconds a client can be idle before the server terminates the session.

options

A ServerOptions enumeration that specifies one or more server options.

Return Value

A boolean value which specifies if the server was started. A return value of **true** specifies that the server has been successfully started. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **Start** method begins listening for client connections on the specified local address and port number. The server is started in its own thread and manages the client sessions independently of the calling thread.

To listen for connections on any suitable IPv4 interface, specify the special dotted-quad address "0.0.0.0". You can accept connections from clients using either IPv4 or IPv6 on the same socket by specifying the special IPv6 address "::0", however this is only supported on Windows 7 SP1 and Windows Server 2008 R2 or later platforms. If no local address is specified, then the server will only listen for connections from clients using IPv4. This behavior is by design for backwards compatibility with systems that do not have an IPv6 TCP/IP stack installed.

It is recommended that you always specify an absolute path for the server root directory, either by passing the full pathname as an argument to this method or by setting the **Directory** property. If the path includes environment variables surrounded by percent (%) symbols, they will be automatically expanded.

If you have configured the server to permit clients to upload files, you must ensure that your application has permission to create files in the directory that you specify. A recommended location for the server root directory would be a subdirectory of the %ALLUSERSPROFILE% directory. Using the environment variable ensures that your server will work correctly on different versions of Windows. If the root directory does not exist at the time that the server is started, it will be created.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.Start Overload List

HttpServer.Start Method (String, Int32, String, Int32, Int32)

Start listening for client connections on the specified IP address and port number.

[Visual Basic]

```
Overloads Public Function Start( _
   ByVal localAddress As String, _
   ByVal localPort As Integer, _
   ByVal rootDirectory As String, _
   ByVal maxClients As Integer, _
   ByVal idleTime As Integer _
) As Boolean
```

[C#]

```
public bool Start(
   string localAddress,
   int localPort,
   string rootDirectory,
   int maxClients,
   int idleTime
);
```

Parameters

localAddress

A string which specifies the local hostname or IP address address that the server should be bound to. If this parameter is an empty string, then an appropriate address will automatically be used. If a specific address is used, the server will only accept client connections on the network interface that is bound to that address.

localPort

An integer that specifies the port number the server should use to listen for client connections. If a value of zero is specified, the server will use the standard port number 80 to listen for connections, or port 443 if the server is configured to use implicit SSL. The port number used by the application must be unique and multiple instances of a server cannot use the same port number. It is recommended that a port number greater than 5000 be used for private, application-specific implementations.

rootDirectory

A string that specifies the path to the root directory for the server. If this value is an empty string, the server will use the current working directory as the root directory.

maxClients

An integer value that specifies the maximum number of clients that may connect to the server.

idleTime

An integer value that specifies the number of seconds a client can be idle before the server terminates the session.

Return Value

A boolean value which specifies if the server was started. A return value of **true** specifies that the server has been successfully started. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **Start** method begins listening for client connections on the specified local address and port number. The server is started in its own thread and manages the client sessions independently of the calling thread.

To listen for connections on any suitable IPv4 interface, specify the special dotted-quad address "0.0.0.0". You can accept connections from clients using either IPv4 or IPv6 on the same socket by specifying the special IPv6 address "::0", however this is only supported on Windows 7 SP1 and Windows Server 2008 R2 or later platforms. If no local address is specified, then the server will only listen for connections from clients using IPv4. This behavior is by design for backwards compatibility with systems that do not have an IPv6 TCP/IP stack installed.

It is recommended that you always specify an absolute path for the server root directory, either by passing the full pathname as an argument to this method or by setting the **Directory** property. If the path includes environment variables surrounded by percent (%) symbols, they will be automatically expanded.

If you have configured the server to permit clients to upload files, you must ensure that your application has permission to create files in the directory that you specify. A recommended location for the server root directory would be a subdirectory of the %ALLUSERSPROFILE% directory. Using the environment variable ensures that your server will work correctly on different versions of Windows. If the root directory does not exist at the time that the server is started, it will be created.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.Start Overload List

HttpServer.Start Method (String, Int32, String, Int32)

Start listening for client connections on the specified IP address and port number.

[Visual Basic]

```
Overloads Public Function Start( _
   ByVal localAddress As String, _
   ByVal localPort As Integer, _
   ByVal rootDirectory As String, _
   ByVal maxClients As Integer _
) As Boolean
```

[C#]

```
public bool Start(
   string localAddress,
   int localPort,
   string rootDirectory,
   int maxClients
);
```

Parameters

localAddress

A string which specifies the local hostname or IP address address that the server should be bound to. If this parameter is an empty string, then an appropriate address will automatically be used. If a specific address is used, the server will only accept client connections on the network interface that is bound to that address.

localPort

An integer that specifies the port number the server should use to listen for client connections. If a value of zero is specified, the server will use the standard port number 80 to listen for connections, or port 443 if the server is configured to use implicit SSL. The port number used by the application must be unique and multiple instances of a server cannot use the same port number. It is recommended that a port number greater than 5000 be used for private, application-specific implementations.

rootDirectory

A string that specifies the path to the root directory for the server. If this value is an empty string, the server will use the current working directory as the root directory.

maxClients

An integer value that specifies the maximum number of clients that may connect to the server.

Return Value

A boolean value which specifies if the server was started. A return value of **true** specifies that the server has been successfully started. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **Start** method begins listening for client connections on the specified local address and port number. The server is started in its own thread and manages the client sessions independently of the calling thread.

To listen for connections on any suitable IPv4 interface, specify the special dotted-quad address "0.0.0.0". You can accept connections from clients using either IPv4 or IPv6 on the same socket by specifying the special IPv6 address "::0", however this is only supported on Windows 7 SP1 and Windows Server 2008 R2 or later platforms. If no local address is specified, then the server will only listen for connections from clients using IPv4. This behavior is by design for backwards compatibility with systems that do not have an

IPv6 TCP/IP stack installed.

It is recommended that you always specify an absolute path for the server root directory, either by passing the full pathname as an argument to this method or by setting the **Directory** property. If the path includes environment variables surrounded by percent (%) symbols, they will be automatically expanded.

If you have configured the server to permit clients to upload files, you must ensure that your application has permission to create files in the directory that you specify. A recommended location for the server root directory would be a subdirectory of the %ALLUSERSPROFILE% directory. Using the environment variable ensures that your server will work correctly on different versions of Windows. If the root directory does not exist at the time that the server is started, it will be created.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.Start Overload List

HttpServer.Start Method (String, Int32, String)

Start listening for client connections on the specified IP address and port number.

[Visual Basic]

```
Overloads Public Function Start( _
   ByVal localAddress As String, _
   ByVal localPort As Integer, _
   ByVal rootDirectory As String _
) As Boolean
```

[C#]

```
public bool Start(
   string localAddress,
   int localPort,
   string rootDirectory
);
```

Parameters

localAddress

A string which specifies the local hostname or IP address address that the server should be bound to. If this parameter is an empty string, then an appropriate address will automatically be used. If a specific address is used, the server will only accept client connections on the network interface that is bound to that address.

localPort

An integer that specifies the port number the server should use to listen for client connections. If a value of zero is specified, the server will use the standard port number 80 to listen for connections, or port 443 if the server is configured to use implicit SSL. The port number used by the application must be unique and multiple instances of a server cannot use the same port number. It is recommended that a port number greater than 5000 be used for private, application-specific implementations.

rootDirectory

A string that specifies the path to the root directory for the server. If this value is an empty string, the server will use the current working directory as the root directory.

Return Value

A boolean value which specifies if the server was started. A return value of **true** specifies that the server has been successfully started. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **Start** method begins listening for client connections on the specified local address and port number. The server is started in its own thread and manages the client sessions independently of the calling thread.

To listen for connections on any suitable IPv4 interface, specify the special dotted-quad address "0.0.0.0". You can accept connections from clients using either IPv4 or IPv6 on the same socket by specifying the special IPv6 address "::0", however this is only supported on Windows 7 SP1 and Windows Server 2008 R2 or later platforms. If no local address is specified, then the server will only listen for connections from clients using IPv4. This behavior is by design for backwards compatibility with systems that do not have an IPv6 TCP/IP stack installed.

It is recommended that you always specify an absolute path for the server root directory, either by passing the full pathname as an argument to this method or by setting the **Directory** property. If the path includes environment variables surrounded by percent (%) symbols, they will be automatically expanded.

If you have configured the server to permit clients to upload files, you must ensure that your application has permission to create files in the directory that you specify. A recommended location for the server root directory would be a subdirectory of the %ALLUSERSPROFILE% directory. Using the environment variable ensures that your server will work correctly on different versions of Windows. If the root directory does not exist at the time that the server is started, it will be created.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.Start Overload List

HttpServer.Start Method (String, Int32)

Start listening for client connections on the specified IP address and port number.

[Visual Basic]

```
Overloads Public Function Start( _
   ByVal localAddress As String, _
   ByVal localPort As Integer _
) As Boolean
```

[C#]

```
public bool Start(
   string localAddress,
   int localPort
);
```

Parameters

localAddress

A string which specifies the local hostname or IP address address that the server should be bound to. If this parameter is an empty string, then an appropriate address will automatically be used. If a specific address is used, the server will only accept client connections on the network interface that is bound to that address.

localPort

An integer that specifies the port number the server should use to listen for client connections. If a value of zero is specified, the server will use the standard port number 80 to listen for connections, or port 443 if the server is configured to use implicit SSL. The port number used by the application must be unique and multiple instances of a server cannot use the same port number. It is recommended that a port number greater than 5000 be used for private, application-specific implementations.

Return Value

A boolean value which specifies if the server was started. A return value of **true** specifies that the server has been successfully started. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **Start** method begins listening for client connections on the specified local address and port number. The server is started in its own thread and manages the client sessions independently of the calling thread.

To listen for connections on any suitable IPv4 interface, specify the special dotted-quad address "0.0.0.0". You can accept connections from clients using either IPv4 or IPv6 on the same socket by specifying the special IPv6 address "::0", however this is only supported on Windows 7 SP1 and Windows Server 2008 R2 or later platforms. If no local address is specified, then the server will only listen for connections from clients using IPv4. This behavior is by design for backwards compatibility with systems that do not have an IPv6 TCP/IP stack installed.

It is recommended that you always specify an absolute path for the server root directory, either by passing the full pathname as an argument to this method or by setting the **Directory** property. If the path includes environment variables surrounded by percent (%) symbols, they will be automatically expanded.

If you have configured the server to permit clients to upload files, you must ensure that your application has permission to create files in the directory that you specify. A recommended location for the server root directory would be a subdirectory of the %ALLUSERSPROFILE% directory. Using the environment variable ensures that your server will work correctly on different versions of Windows. If the root directory does not exist at the time that the server is started, it will be created.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.Start Overload List

HttpServer.Start Method (Int32)

Start listening for client connections on the specified IP address and port number.

[Visual Basic]

```
Overloads Public Function Start( _
   ByVal localPort As Integer _
) As Boolean
```

[C#]

```
public bool Start(
   int localPort
);
```

Parameters

localPort

An integer that specifies the port number the server should use to listen for client connections. If a value of zero is specified, the server will use the standard port number 80 to listen for connections, or port 443 if the server is configured to use implicit SSL. The port number used by the application must be unique and multiple instances of a server cannot use the same port number. It is recommended that a port number greater than 5000 be used for private, application-specific implementations.

Return Value

A boolean value which specifies if the server was started. A return value of **true** specifies that the server has been successfully started. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **Start** method begins listening for client connections on the specified local address and port number. The server is started in its own thread and manages the client sessions independently of the calling thread.

To listen for connections on any suitable IPv4 interface, specify the special dotted-quad address "0.0.0.0". You can accept connections from clients using either IPv4 or IPv6 on the same socket by specifying the special IPv6 address "::0", however this is only supported on Windows 7 SP1 and Windows Server 2008 R2 or later platforms. If no local address is specified, then the server will only listen for connections from clients using IPv4. This behavior is by design for backwards compatibility with systems that do not have an IPv6 TCP/IP stack installed.

It is recommended that you always specify an absolute path for the server root directory, either by passing the full pathname as an argument to this method or by setting the **Directory** property. If the path includes environment variables surrounded by percent (%) symbols, they will be automatically expanded.

If you have configured the server to permit clients to upload files, you must ensure that your application has permission to create files in the directory that you specify. A recommended location for the server root directory would be a subdirectory of the %ALLUSERSPROFILE% directory. Using the environment variable ensures that your server will work correctly on different versions of Windows. If the root directory does not exist at the time that the server is started, it will be created.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.Start Overload List

HttpServer.Start Method ()

Start listening for client connections on the specified IP address and port number.

[Visual Basic]

```
Overloads Public Function Start() As Boolean
```

[C#]

```
public bool Start();
```

Return Value

A boolean value which specifies if the server was started. A return value of **true** specifies that the server has been successfully started. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure

Remarks

The **Start** method begins listening for client connections on the specified local address and port number. The server is started in its own thread and manages the client sessions independently of the calling thread.

To listen for connections on any suitable IPv4 interface, specify the special dotted-quad address "0.0.0.0". You can accept connections from clients using either IPv4 or IPv6 on the same socket by specifying the special IPv6 address "::0", however this is only supported on Windows 7 SP1 and Windows Server 2008 R2 or later platforms. If no local address is specified, then the server will only listen for connections from clients using IPv4. This behavior is by design for backwards compatibility with systems that do not have an IPv6 TCP/IP stack installed.

It is recommended that you always specify an absolute path for the server root directory, either by passing the full pathname as an argument to this method or by setting the **Directory** property. If the path includes environment variables surrounded by percent (%) symbols, they will be automatically expanded.

If you have configured the server to permit clients to upload files, you must ensure that your application has permission to create files in the directory that you specify. A recommended location for the server root directory would be a subdirectory of the %ALLUSERSPROFILE% directory. Using the environment variable ensures that your server will work correctly on different versions of Windows. If the root directory does not exist at the time that the server is started, it will be created.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.Start Overload List

HttpServer.Stop Method

Stop listening for new client connections and terminate all active clients already connected to the server.

[Visual Basic]

```
Public Function Stop() As Boolean
```

[C#]

```
public bool Stop();
```

Return Value

A boolean value which specifies if the server was stopped. A return value of **true** specifies that the server has been successfully stopped. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Stop** method instructs the server to stop accepting client connections, disconnects all active client connections and terminates the thread that is managing the server session. If this method is called when there is one or more clients connected to the server, it will signal each client thread to terminate and then wait for the server thread to terminate.

See Also

HttpServer Class | SocketTools Namespace | Restart Method | Resume Method | Start Method | Throttle Method

HttpServer.Suspend Method

Suspend accepting new client connections.

[Visual Basic]
```
Public Function Suspend() As Boolean
```

[C#]
```
public bool Suspend();
```

Return Value

A boolean value which specifies if the server has been suspended. A return value of **true** specifies that the server has been successfully stopped. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Suspend** method instructs the server to suspend accepting new client connections. All new clients that attempt to connect to the server will be will be immediately rejected by the server. To resume accepting new client connections, call the **Resume** method. This method will not affect those clients that have already established a connection with the server before the **Suspend** method was called.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.Throttle Method

Limit the maximum number of client connections.

Overload List

Limit the maximum number of client connections.

public bool Throttle(int);

Limit the maximum number of client connections and connections per IP address.

public bool Throttle(int,int);

Limit the maximum number of client connections, connections per IP address and connection rate.

public bool Throttle(int,int,int);

See Also

HttpServer Class | SocketTools Namespace

HttpServer.Throttle Method (Int32, Int32, Int32)

Limit the maximum number of client connections, connections per IP address and connection rate.

[Visual Basic]

```
Overloads Public Function Throttle( _
   ByVal maxClients As Integer, _
   ByVal maxClientsPerAddress As Integer, _
   ByVal connectionRate As Integer _
) As Boolean
```

[C#]

```
public bool Throttle(
   int maxClients,
   int maxClientsPerAddress,
   int connectionRate
);
```

Parameters

maxClients

An integer value that specifies the maximum number of clients that may connect to the server. A value of zero specifies that there is no fixed limit to the number of client connections.

maxClientsPerAddress

An integer value that specifies the maximum number of clients that may connect to the server from the same IP address. A value of zero specifies that there is no fixed limit to the number of client connections per address. By default, there is no limit on the number of client connections per address.

connectionRate

An integer value that specifies a restriction on the rate of client connections, limiting the number of connections that will be accepted within that period of time. A value of zero specifies that there is no restriction on the rate of client connections. The higher this value, the fewer the number of connections that will be accepted within a specific period of time. By default, there is no limit on the client connection rate.

Return Value

A boolean value which specifies if the method was successful. A return value of **true** indicates success. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

If the maximum number of client connections or maximum number of connections per address is exceeded, the server will reject subsequent connection attempts until the number of active client sessions drops below the specified threshold. Note that adjusting these values lower than the current connection limits will not affect clients that have already connected to the server. For example, if the **Start** method is called with the maximum number of clients set to 100, and then the **Throttle** method is called lowering that value to 75, no existing client connections will be affected by the change. However, the server will not accept any new connections until the number of active clients drops below 75.

If the value of the ***maxGuests*** parameter is greater than zero, then anonymous logins will be enabled and clients can authenticate with the username "anonymous" and their email address as the password. If the parameter is set to zero, then anonymous logins will be disabled. Note that this will not affect any clients that are currently logged in, it only affects those clients that connect after the **Throttle** method has been called.

Increasing the connection rate value will force the server to slow down the rate at which it will accept incoming client connection requests. For example, setting this parameter to a value of 1000 would limit the server to accepting one client connection every second, while a value of 250 would allow the server to accept four client connections per second. Note that significantly increasing the amount of time the server must wait to accept client connections can exceed the connection backlog queue, resulting in client connections being rejected.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.Throttle Overload List

HttpServer.Throttle Method (Int32, Int32)

Limit the maximum number of client connections and connections per IP address.

[Visual Basic]

```
Overloads Public Function Throttle( _
   ByVal maxClients As Integer, _
   ByVal maxClientsPerAddress As Integer _
) As Boolean
```

[C#]

```
public bool Throttle(
   int maxClients,
   int maxClientsPerAddress
);
```

Parameters

maxClients

An integer value that specifies the maximum number of clients that may connect to the server. A value of zero specifies that there is no fixed limit to the number of client connections.

maxClientsPerAddress

An integer value that specifies the maximum number of clients that may connect to the server from the same IP address. A value of zero specifies that there is no fixed limit to the number of client connections per address. By default, there is no limit on the number of client connections per address.

Return Value

A boolean value which specifies if the method was successful. A return value of **true** indicates success. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

If the maximum number of client connections or maximum number of connections per address is exceeded, the server will reject subsequent connection attempts until the number of active client sessions drops below the specified threshold. Note that adjusting these values lower than the current connection limits will not affect clients that have already connected to the server. For example, if the **Start** method is called with the maximum number of clients set to 100, and then the **Throttle** method is called lowering that value to 75, no existing client connections will be affected by the change. However, the server will not accept any new connections until the number of active clients drops below 75.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.Throttle Overload List

HttpServer.Throttle Method (Int32)

Limit the maximum number of client connections.

[Visual Basic]

```
Overloads Public Function Throttle( _
   ByVal maxClients As Integer _
) As Boolean
```

[C#]

```
public bool Throttle(
   int maxClients
);
```

Parameters

maxClients

An integer value that specifies the maximum number of clients that may connect to the server. A value of zero specifies that there is no fixed limit to the number of client connections.

Return Value

A boolean value which specifies if the method was successful. A return value of **true** indicates success. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

If the maximum number of client connections or maximum number of connections per address is exceeded, the server will reject subsequent connection attempts until the number of active client sessions drops below the specified threshold. Note that adjusting these values lower than the current connection limits will not affect clients that have already connected to the server. For example, if the **Start** method is called with the maximum number of clients set to 100, and then the **Throttle** method is called lowering that value to 75, no existing client connections will be affected by the change. However, the server will not accept any new connections until the number of active clients drops below 75.

See Also

HttpServer Class | SocketTools Namespace | HttpServer.Throttle Overload List

HttpServer.Uninitialize Method

Uninitialize the class library and release any resources allocated for the server.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method terminates any active connection, releases resources allocated for the server and unloads the networking library. After this method has been called, no further network operations may be performed until the class instance has been re-initialized.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the class is no longer needed.

See Also

HttpServer Class | SocketTools Namespace | Initialize Method

HttpServer Events

The events of the **HttpServer** class are listed below. For a complete list of **HttpServer** class members, see the HttpServer Members topic.

Public Instance Events

⚡ OnAuthenticate	Occurs when the client has requested authentication with the specified username and password.
⚡ OnCommand	Occurs when a client has issued a command to the server.
⚡ OnConnect	Occurs when a connection is established with the remote host.
⚡ OnDisconnect	Occurs when the remote host disconnects from the local system.
⚡ OnDownload	Occurs when a connection is established with the remote host.
⚡ OnError	Occurs when an network operation fails.
⚡ OnExecute	Occurs when the client has executed an external script handler on the server.
⚡ OnIdle	Occurs when the there are no clients connected to the server.
⚡ OnResult	Occurs when the command issued by the client has been processed by the server.
⚡ OnStart	Occurs when the server starts accepting connections.
⚡ OnStop	Occurs when the server stops accepting connections.
⚡ OnTimeout	Occurs when the client has exceeded the maximum allowed idle time.
⚡ OnUpload	Occurs when the client has successfully uploaded a file to the server.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.OnAuthenticate Event

Occurs when the client has requested authentication with the specified username and password.

[Visual Basic]

```
Public Event OnAuthenticate As OnAuthenticateEventHandler
```

[C#]

```
public event OnAuthenticateEventHandler OnAuthenticate;
```

Event Data

The event handler receives an argument of type HttpServer.AuthenticateEventArgs containing data related to this event. The following **HttpServer.AuthenticateEventArgs** properties provide information specific to this event.

Property	Description
ClientId	Gets a value that uniquely identifies the client session.
HostName	Gets the host name used by the client to establish the connection to the server.
Password	Gets the password provided by the client for authentication.
UserName	Gets the password provided by the client for authentication.

Remarks

The **OnAuthenticate** event occurs when the client has requested authentication by sending the USER and PASS command to the server. The event handler can call the **Authenticate** method to authenticate the client session. If the client is not authenticated, the server will send an error message to the client and terminate the session.

If the application has created one or more virtual users using the **AddUser** method and/or the **LocalUser** property has been set to **True**, it is not necessary to implement an **OnAuthenticate** handler unless you also wish to perform custom authentication for specific users.

This event handler will be invoked in the context of the worker thread that is managing the client session, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.AuthenticateEventArgs Class

Provides data for the OnAuthenticate event.

For a list of all members of this type, see HttpServer.AuthenticateEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.HttpServer.AuthenticateEventArgs

[Visual Basic]

```
Public Class HttpServer.AuthenticateEventArgs
    Inherits EventArgs
```

[C#]

```
public class HttpServer.AuthenticateEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

HttpServer.AuthenticateEventArgs Members | SocketTools Namespace

HttpServer.AuthenticateEventArgs Constructor

Initializes a new instance of the HttpServer.AuthenticateEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public HttpServer.AuthenticateEventArgs();
```

See Also

HttpServer.AuthenticateEventArgs Class | SocketTools Namespace

HttpServer.AuthenticateEventArgs Members

HttpServer.AuthenticateEventArgs overview

Public Instance Constructors

HttpServer.AuthenticateEventArgs Constructor	Initializes a new instance of the HttpServer.AuthenticateEventArgs class.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
HostName	Gets the host name used by the client to establish the connection to the server.
Password	Gets the password provided by the client for authentication.
UserName	Gets the password provided by the client for authentication.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpServer.AuthenticateEventArgs Class | SocketTools Namespace

HttpServer.AuthenticateEventArgs Properties

The properties of the **HttpServer.AuthenticateEventArgs** class are listed below. For a complete list of **HttpServer.AuthenticateEventArgs** class members, see the HttpServer.AuthenticateEventArgs Members topic.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
HostName	Gets the host name used by the client to establish the connection to the server.
Password	Gets the password provided by the client for authentication.
UserName	Gets the password provided by the client for authentication.

See Also

HttpServer.AuthenticateEventArgs Class | SocketTools Namespace

HttpServer.AuthenticateEventArgs.ClientId Property

Gets a value that uniquely identifies the client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which identifies the client session.

Remarks

The **ClientId** property returns a unique integer value that identifies the client session that generated the event.

See Also

HttpServer.AuthenticateEventArgs Class | SocketTools Namespace

HttpServer.AuthenticateEventArgs.HostName Property

Gets the host name used by the client to establish the connection to the server.

[Visual Basic]

```
Public ReadOnly Property HostName As String
```

[C#]

```
public string HostName {get;}
```

Property Value

A string that specifies a host name.

See Also

HttpServer.AuthenticateEventArgs Class | SocketTools Namespace

HttpServer.AuthenticateEventArgs.Password Property

Gets the password provided by the client for authentication.

[Visual Basic]

```
Public ReadOnly Property Password As String
```

[C#]

```
public string Password {get;}
```

Property Value

A string that specifies the password provided by the client when it requests authentication.

See Also

HttpServer.AuthenticateEventArgs Class | SocketTools Namespace

HttpServer.AuthenticateEventArgs.UserName Property

Gets the password provided by the client for authentication.

[Visual Basic]

```
Public ReadOnly Property UserName As String
```

[C#]

```
public string UserName {get;}
```

Property Value

A string that specifies the username provided by the client when it requests authentication.

See Also

HttpServer.AuthenticateEventArgs Class | SocketTools Namespace

HttpServer.OnCommand Event

Occurs when a client has issued a command to the server.

[Visual Basic]

```
Public Event OnCommand As OnCommandEventHandler
```

[C#]

```
public event OnCommandEventHandler OnCommand;
```

Event Data

The event handler receives an argument of type HttpServer.CommandEventArgs containing data related to this event. The following **HttpServer.CommandEventArgs** properties provide information specific to this event.

Property	Description
ClientId	Gets a value that uniquely identifies the client session.
Command	Gets a value that specifies the command issued by the client.
Parameters	Gets a value that specifies the command parameters issued by the client.
Resource	Gets a value that specifies the resource requested by the client.

Remarks

The **OnCommand** event occurs after the client has sent a command to the server, but before the command has been processed. This event occurs for all commands issued by the client, including invalid or disabled commands. If the application wishes to handle the command itself, it must perform any processing and then call the **SendResponse** method to send the success or error code to the client. If the **SendResponse** method is not called, then the server will perform its default processing for the command.

After the command has been processed, the **OnResult** event handler will be invoked.

This event handler will be invoked in the context of the worker thread that is managing the client session, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.CommandEventArgs Class

Provides data for the OnCommand event.

For a list of all members of this type, see HttpServer.CommandEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.HttpServer.CommandEventArgs

[Visual Basic]
```
Public Class HttpServer.CommandEventArgs
    Inherits EventArgs
```

[C#]
```
public class HttpServer.CommandEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

HttpServer.CommandEventArgs Members | SocketTools Namespace

HttpServer.CommandEventArgs Constructor

Initializes a new instance of the HttpServer.CommandEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public HttpServer.CommandEventArgs();
```

See Also

HttpServer.CommandEventArgs Class | SocketTools Namespace

HttpServer.CommandEventArgs Members

HttpServer.CommandEventArgs overview

Public Instance Constructors

HttpServer.CommandEventArgs Constructor	Initializes a new instance of the HttpServer.CommandEventArgs class.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
Command	Gets a value that specifies the command issued by the client.
Parameters	Gets a value that specifies the command parameters issued by the client.
Resource	Gets a value that specifies the resource requested by the client.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpServer.CommandEventArgs Class | SocketTools Namespace

HttpServer.CommandEventArgs Properties

The properties of the **HttpServer.CommandEventArgs** class are listed below. For a complete list of **HttpServer.CommandEventArgs** class members, see the HttpServer.CommandEventArgs Members topic.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
Command	Gets a value that specifies the command issued by the client.
Parameters	Gets a value that specifies the command parameters issued by the client.
Resource	Gets a value that specifies the resource requested by the client.

See Also

HttpServer.CommandEventArgs Class | SocketTools Namespace

HttpServer.CommandEventArgs.ClientId Property

Gets a value that uniquely identifies the client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which identifies the client session.

Remarks

The **ClientId** property returns a unique integer value that identifies the client session that generated the event.

See Also

HttpServer.CommandEventArgs Class | SocketTools Namespace

HttpServer.CommandEventArgs.Command Property

Gets a value that specifies the command issued by the client.

[Visual Basic]

```
Public ReadOnly Property Command As String
```

[C#]

```
public string Command {get;}
```

Property Value

A string that specifies the command sent by the client.

See Also

HttpServer.CommandEventArgs Class | SocketTools Namespace

HttpServer.CommandEventArgs.Parameters Property

Gets a value that specifies the command parameters issued by the client.

[Visual Basic]
```
Public ReadOnly Property Parameters As String
```

[C#]
```
public string Parameters {get;}
```

Property Value

A string that specifies the command parameters sent by the client.

See Also

HttpServer.CommandEventArgs Class | SocketTools Namespace

HttpServer.CommandEventArgs.Resource Property

Gets a value that specifies the resource requested by the client.

[Visual Basic]

```
Public ReadOnly Property Resource As String
```

[C#]

```
public string Resource {get;}
```

Property Value

A string value that specifies the resource that the client has requested. Depending on the command issued, it may be a document, a folder or an executable script.

See Also

HttpServer.CommandEventArgs Class | SocketTools Namespace

HttpServer.OnConnect Event

Occurs when a connection is established with the remote host.

[Visual Basic]

```
Public Event OnConnect As OnConnectEventHandler
```

[C#]

```
public event OnConnectEventHandler OnConnect;
```

Event Data

The event handler receives an argument of type HttpServer.ConnectEventArgs containing data related to this event. The following **HttpServer.ConnectEventArgs** properties provide information specific to this event.

Property	Description
ClientAddress	Gets the address of the client establishing the connection.
ClientId	Gets a value that uniquely identifies the client session.

Remarks

The **OnConnect** event occurs after the client has established its initial connection to the server, after the server has checked the active client limits and the TLS handshake has been performed if required. If the server has been suspended, or the limit on the maximum number of client sessions has been exceeded, the server will terminate the client session prior to this event handler being invoked.

This event handler will be invoked in the context of the worker thread that is managing the client session, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.ConnectEventArgs Class

Provides data for the OnConnect event.

For a list of all members of this type, see HttpServer.ConnectEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.HttpServer.ConnectEventArgs

[Visual Basic]

```
Public Class HttpServer.ConnectEventArgs
    Inherits EventArgs
```

[C#]

```
public class HttpServer.ConnectEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

HttpServer.ConnectEventArgs Members | SocketTools Namespace | OnConnect Event (SocketTools.HttpServer)

HttpServer.ConnectEventArgs Constructor

Initializes a new instance of the HttpServer.ConnectEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public HttpServer.ConnectEventArgs();
```

See Also

HttpServer.ConnectEventArgs Class | SocketTools Namespace

HttpServer.ConnectEventArgs Members

HttpServer.ConnectEventArgs overview

Public Instance Constructors

HttpServer.ConnectEventArgs Constructor	Initializes a new instance of the HttpServer.ConnectEventArgs class.

Public Instance Properties

ClientAddress	Gets the address of the client establishing the connection.
ClientId	Gets a value that uniquely identifies the client session.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpServer.ConnectEventArgs Class | SocketTools Namespace | OnConnect Event (SocketTools.HttpServer)

HttpServer.ConnectEventArgs Properties

The properties of the **HttpServer.ConnectEventArgs** class are listed below. For a complete list of **HttpServer.ConnectEventArgs** class members, see the HttpServer.ConnectEventArgs Members topic.

Public Instance Properties

ClientAddress	Gets the address of the client establishing the connection.
ClientId	Gets a value that uniquely identifies the client session.

See Also

HttpServer.ConnectEventArgs Class | SocketTools Namespace | OnConnect Event (SocketTools.HttpServer)

HttpServer.ConnectEventArgs.ClientAddress Property

Gets the address of the client establishing the connection.

[Visual Basic]

```
Public ReadOnly Property ClientAddress As String
```

[C#]

```
public string ClientAddress {get;}
```

Property Value

A string that specifies the Internet Protocol address of the client.

See Also

HttpServer.ConnectEventArgs Class | SocketTools Namespace

HttpServer.ConnectEventArgs.ClientId Property

Gets a value that uniquely identifies the client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which identifies the client session.

Remarks

The **ClientId** property returns a unique integer value that identifies the client session that generated the event.

See Also

HttpServer.ConnectEventArgs Class | SocketTools Namespace

HttpServer.OnDownload Event

Occurs when a connection is established with the remote host.

[Visual Basic]
```
Public Event OnDownload As OnDownloadEventHandler
```

[C#]
```
public event OnDownloadEventHandler OnDownload;
```

Event Data

The event handler receives an argument of type HttpServer.DownloadEventArgs containing data related to this event. The following **HttpServer.DownloadEventArgs** properties provide information specific to this event.

Property	Description
ClientId	Gets a value that uniquely identifies the client session.
FileName	Gets a value that specifies the file being downloaded.
FileSize	Gets a value that specifies the size of the file.

Remarks

The **OnDownload** event occurs after the client has successfully downloaded a file from the server using the GET command. If the file transfer fails or is aborted, this event will not occur.

This event handler will be invoked in the context of the worker thread that is managing the client session, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.DownloadEventArgs Class

Provides data for the OnDownload event.

For a list of all members of this type, see HttpServer.DownloadEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.HttpServer.DownloadEventArgs

[Visual Basic]

```
Public Class HttpServer.DownloadEventArgs
    Inherits EventArgs
```

[C#]

```
public class HttpServer.DownloadEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

HttpServer.DownloadEventArgs Members | SocketTools Namespace

HttpServer.DownloadEventArgs Constructor

Initializes a new instance of the HttpServer.DownloadEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public HttpServer.DownloadEventArgs();
```

See Also

HttpServer.DownloadEventArgs Class | SocketTools Namespace

HttpServer.DownloadEventArgs Members

HttpServer.DownloadEventArgs overview

Public Instance Constructors

HttpServer.DownloadEventArgs Constructor	Initializes a new instance of the HttpServer.DownloadEventArgs class.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
FileName	Gets a value that specifies the file being downloaded.
FileSize	Gets a value that specifies the size of the file.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpServer.DownloadEventArgs Class | SocketTools Namespace

HttpServer.DownloadEventArgs Properties

The properties of the **HttpServer.DownloadEventArgs** class are listed below. For a complete list of **HttpServer.DownloadEventArgs** class members, see the HttpServer.DownloadEventArgs Members topic.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
FileName	Gets a value that specifies the file being downloaded.
FileSize	Gets a value that specifies the size of the file.

See Also

HttpServer.DownloadEventArgs Class | SocketTools Namespace

HttpServer.DownloadEventArgs.ClientId Property

Gets a value that uniquely identifies the client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which identifies the client session.

Remarks

The **ClientId** property returns a unique integer value that identifies the client session that generated the event.

See Also

HttpServer.DownloadEventArgs Class | SocketTools Namespace

HttpServer.DownloadEventArgs.FileName Property

Gets a value that specifies the file being downloaded.

[Visual Basic]

```
Public ReadOnly Property FileName As String
```

[C#]

```
public string FileName {get;}
```

Property Value

A string that specifies the full path to a file on the local system.

See Also

HttpServer.DownloadEventArgs Class | SocketTools Namespace

HttpServer.DownloadEventArgs.FileSize Property

Gets a value that specifies the size of the file.

[Visual Basic]

```
Public ReadOnly Property FileSize As Long
```

[C#]

```
public long FileSize {get;}
```

Property Value

A long integer value that specifies the size of the file in bytes.

See Also

HttpServer.DownloadEventArgs Class | SocketTools Namespace

HttpServer.OnDisconnect Event

Occurs when the remote host disconnects from the local system.

[Visual Basic]
```
Public Event OnDisconnect As OnDisconnectEventHandler
```

[C#]
```
public event OnDisconnectEventHandler OnDisconnect;
```

Event Data

The event handler receives an argument of type HttpServer.DisconnectEventArgs containing data related to this event. The following **HttpServer.DisconnectEventArgs** property provides information specific to this event.

Property	Description
ClientId	Gets a value that uniquely identifies the client session.

Remarks

The **OnDisconnect** event is generated when the connection is terminated by the client and there is no more data available to be read.

It is not necessary to call the **Disconnect** method inside the **OnDisconnect** event handler because the client session is already in the process of disconnecting from the server.

This event handler will be invoked in the context of the worker thread that is managing the client session, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.DisconnectEventArgs Class

Provides data for the OnDisconnect event.

For a list of all members of this type, see HttpServer.DisconnectEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.HttpServer.DisconnectEventArgs

[Visual Basic]
```
Public Class HttpServer.DisconnectEventArgs
    Inherits EventArgs
```

[C#]
```
public class HttpServer.DisconnectEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

HttpServer.DisconnectEventArgs Members | SocketTools Namespace | OnDisconnect Event (SocketTools.HttpServer)

HttpServer.DisconnectEventArgs Constructor

Initializes a new instance of the HttpServer.DisconnectEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public HttpServer.DisconnectEventArgs();
```

See Also

HttpServer.DisconnectEventArgs Class | SocketTools Namespace

HttpServer.DisconnectEventArgs Members

HttpServer.DisconnectEventArgs overview

Public Instance Constructors

HttpServer.DisconnectEventArgs Constructor	Initializes a new instance of the HttpServer.DisconnectEventArgs class.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpServer.DisconnectEventArgs Class | SocketTools Namespace | OnDisconnect Event (SocketTools.HttpServer)

HttpServer.DisconnectEventArgs Properties

The properties of the **HttpServer.DisconnectEventArgs** class are listed below. For a complete list of **HttpServer.DisconnectEventArgs** class members, see the HttpServer.DisconnectEventArgs Members topic.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.

See Also

HttpServer.DisconnectEventArgs Class | SocketTools Namespace | OnDisconnect Event (SocketTools.HttpServer)

HttpServer.DisconnectEventArgs.ClientId Property

Gets a value that uniquely identifies the client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which identifies the client session.

Remarks

The **ClientId** property returns a unique integer value that identifies the client session that generated the event.

See Also

HttpServer.DisconnectEventArgs Class | SocketTools Namespace

HttpServer.OnError Event

Occurs when an network operation fails.

[Visual Basic]
```
Public Event OnError As OnErrorEventHandler
```

[C#]
```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type HttpServer.ErrorEventArgs containing data related to this event. The following **HttpServer.ErrorEventArgs** properties provide information specific to this event.

Property	Description
ClientId	Gets a value that uniquely identifies the client session.
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a network operation fails.

This event handler may be invoked in the context of the worker thread that is managing the client session, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see HttpServer.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.HttpServer.ErrorEventArgs

[Visual Basic]
```
Public Class HttpServer.ErrorEventArgs
    Inherits EventArgs
```

[C#]
```
public class HttpServer.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

HttpServer.ErrorEventArgs Members | SocketTools Namespace

HttpServer.ErrorEventArgs Constructor

Initializes a new instance of the HttpServer.ErrorEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public HttpServer.ErrorEventArgs();
```

See Also

HttpServer.ErrorEventArgs Class | SocketTools Namespace

HttpServer.ErrorEventArgs Members

HttpServer.ErrorEventArgs overview

Public Instance Constructors

HttpServer.ErrorEventArgs Constructor	Initializes a new instance of the HttpServer.ErrorEventArgs class.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpServer.ErrorEventArgs Class | SocketTools Namespace

HttpServer.ErrorEventArgs Properties

The properties of the **HttpServer.ErrorEventArgs** class are listed below. For a complete list of **HttpServer.ErrorEventArgs** class members, see the HttpServer.ErrorEventArgs Members topic.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

HttpServer.ErrorEventArgs Class | SocketTools Namespace

HttpServer.ErrorEventArgs.ClientId Property

Gets a value that uniquely identifies the client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which identifies the client session.

Remarks

The **ClientId** property returns a unique integer value that identifies the client session that generated the event.

See Also

HttpServer.ErrorEventArgs Class | SocketTools Namespace

HttpServer.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

HttpServer.ErrorEventArgs Class | SocketTools Namespace | Error Property

HttpServer.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public HttpServer.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

HttpServer.ErrorEventArgs Class | SocketTools Namespace | Description Property

HttpServer.OnExecute Event

Occurs when the client has executed an external script handler on the server.

[Visual Basic]

```
Public Event OnExecute As OnExecuteEventHandler
```

[C#]

```
public event OnExecuteEventHandler OnExecute;
```

Event Data

The event handler receives an argument of type HttpServer.ExecuteEventArgs containing data related to this event. The following **HttpServer.ExecuteEventArgs** properties provide information specific to this event.

Property	Description
ClientId	Gets a value that uniquely identifies the client session.
ExitCode	Gets a value that specifies the exit code for the script handler.
Output	Gets the output of the script handler executed by the client.
Parameters	Gets a value that specifies the parameters included in the request.
Resource	Gets a value that specifies the resource requested by the client.

Remarks

The **OnExecute** event occurs after the client has successfully executed an external script handler.

This event will only be generated if the client has the **httpAccessExecute** permission. Clients are not granted this permission by default, and must be explicitly permitted to execute external programs. If the client does have this permission, it can only execute specific programs that have been registered by the server application using the **RegisterProgram** method.

This event handler will be invoked in the context of the worker thread that is managing the client session, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.ExecuteEventArgs Class

Provides data for the OnExecute event.

For a list of all members of this type, see HttpServer.ExecuteEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.HttpServer.ExecuteEventArgs

[Visual Basic]
```
Public Class HttpServer.ExecuteEventArgs
    Inherits EventArgs
```

[C#]
```
public class HttpServer.ExecuteEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

HttpServer.ExecuteEventArgs Members | SocketTools Namespace

HttpServer.ExecuteEventArgs Constructor

Initializes a new instance of the HttpServer.ExecuteEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public HttpServer.ExecuteEventArgs();
```

See Also

HttpServer.ExecuteEventArgs Class | SocketTools Namespace

HttpServer.ExecuteEventArgs Members

HttpServer.ExecuteEventArgs overview

Public Instance Constructors

HttpServer.ExecuteEventArgs Constructor	Initializes a new instance of the HttpServer.ExecuteEventArgs class.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
ExitCode	Gets a value that specifies the exit code for the script handler.
Output	Gets the output of the script handler executed by the client.
Parameters	Gets a value that specifies the parameters included in the request.
Resource	Gets a value that specifies the resource requested by the client.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpServer.ExecuteEventArgs Class | SocketTools Namespace

HttpServer.ExecuteEventArgs Properties

The properties of the **HttpServer.ExecuteEventArgs** class are listed below. For a complete list of **HttpServer.ExecuteEventArgs** class members, see the HttpServer.ExecuteEventArgs Members topic.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
ExitCode	Gets a value that specifies the exit code for the script handler.
Output	Gets the output of the script handler executed by the client.
Parameters	Gets a value that specifies the parameters included in the request.
Resource	Gets a value that specifies the resource requested by the client.

See Also

HttpServer.ExecuteEventArgs Class | SocketTools Namespace

HttpServer.ExecuteEventArgs.ClientId Property

Gets a value that uniquely identifies the client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which identifies the client session.

Remarks

The **ClientId** property returns a unique integer value that identifies the client session that generated the event.

See Also

HttpServer.ExecuteEventArgs Class | SocketTools Namespace

HttpServer.ExecuteEventArgs.ExitCode Property

Gets a value that specifies the exit code for the script handler.

[Visual Basic]

```
Public ReadOnly Property ExitCode As Integer
```

[C#]

```
public int ExitCode {get;}
```

Property Value

An integer value that specifies an exit code.

See Also

HttpServer.ExecuteEventArgs Class | SocketTools Namespace

HttpServer.ExecuteEventArgs.Output Property

Gets the output of the script handler executed by the client.

[Visual Basic]

```
Public ReadOnly Property Output As String
```

[C#]

```
public string Output {get;}
```

Property Value

A string that contains the output of the script handler executed on behalf of the client.

See Also

HttpServer.ExecuteEventArgs Class | SocketTools Namespace

HttpServer.ExecuteEventArgs.Resource Property

Gets a value that specifies the resource requested by the client.

[Visual Basic]

```
Public ReadOnly Property Resource As String
```

[C#]

```
public string Resource {get;}
```

Property Value

A string that specifies the virtual path to the script or executable that was requested by the client.

See Also

HttpServer.ExecuteEventArgs Class | SocketTools Namespace

HttpServer.OnIdle Event

Occurs when the there are no clients connected to the server.

[Visual Basic]

```
Public Event OnIdle As EventHandler
```

[C#]

```
public event EventHandler OnIdle;
```

Remarks

This event will only occur after at least one client has connected to the server and then closes its connection or is disconnected. This event will not occur immediately after the server has started using the **Start** method, and will not occur when the server is stopped using the **Stop** method. Your application should implement an **OnStart** event handler for when the server first starts, and an **OnStop** event handler for when the server is stopped.

If one or more new client connections are accepted after this event occurs, the event will be generated again when those clients disconnect and the active client count drops to zero. Therefore it is to be expected that this event will occur multiple times over the lifetime of the server as it continues to listen for connections

This event handler will be invoked in the context of the worker thread that is managing the server, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.OnResult Event

Occurs when the command issued by the client has been processed by the server.

[Visual Basic]
```
Public Event OnResult As OnResultEventHandler
```

[C#]
```
public event OnResultEventHandler OnResult;
```

Event Data

The event handler receives an argument of type HttpServer.ResultEventArgs containing data related to this event. The following **HttpServer.ResultEventArgs** properties provide information specific to this event.

Property	Description
ClientId	Gets a value that uniquely identifies the client session.
Command	Gets a value that specifies the command issued by the client.
Resource	Gets a value that specifies the resource requested by the client.
ResultCode	Gets the result code associated with the last command issued by the client.

Remarks

The **OnResult** event occurs after the server has processed a command issued by the client. This event will inform the application whether the command that was issued by the client was successful or not. If the command was successful, then other related events such as **OnDownload** may also fire after this event.

This event handler will be invoked in the context of the worker thread that is managing the client session, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.ResultEventArgs Class

Provides data for the OnResult event.

For a list of all members of this type, see HttpServer.ResultEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.HttpServer.ResultEventArgs

[Visual Basic]
```
Public Class HttpServer.ResultEventArgs
    Inherits EventArgs
```

[C#]
```
public class HttpServer.ResultEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

HttpServer.ResultEventArgs Members | SocketTools Namespace

HttpServer.ResultEventArgs Constructor

Initializes a new instance of the HttpServer.ResultEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public HttpServer.ResultEventArgs();
```

See Also

HttpServer.ResultEventArgs Class | SocketTools Namespace

HttpServer.ResultEventArgs Members

HttpServer.ResultEventArgs overview

Public Instance Constructors

HttpServer.ResultEventArgs Constructor	Initializes a new instance of the HttpServer.ResultEventArgs class.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
Command	Gets a value that specifies the command issued by the client.
Resource	Gets a value that specifies the resource requested by the client.
ResultCode	Gets the result code associated with the last command issued by the client.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpServer.ResultEventArgs Class | SocketTools Namespace

HttpServer.ResultEventArgs Properties

The properties of the **HttpServer.ResultEventArgs** class are listed below. For a complete list of **HttpServer.ResultEventArgs** class members, see the HttpServer.ResultEventArgs Members topic.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
Command	Gets a value that specifies the command issued by the client.
Resource	Gets a value that specifies the resource requested by the client.
ResultCode	Gets the result code associated with the last command issued by the client.

See Also

HttpServer.ResultEventArgs Class | SocketTools Namespace

HttpServer.ResultEventArgs.ClientId Property

Gets a value that uniquely identifies the client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which identifies the client session.

Remarks

The **ClientId** property returns a unique integer value that identifies the client session that generated the event.

See Also

HttpServer.ResultEventArgs Class | SocketTools Namespace

HttpServer.ResultEventArgs.Command Property

Gets a value that specifies the command issued by the client.

[Visual Basic]

```
Public ReadOnly Property Command As String
```

[C#]

```
public string Command {get;}
```

Property Value

A string that specifies the command sent by the client.

See Also

HttpServer.ResultEventArgs Class | SocketTools Namespace

HttpServer.ResultEventArgs.ResultCode Property

Gets the result code associated with the last command issued by the client.

[Visual Basic]

```
Public ReadOnly Property ResultCode As Integer
```

[C#]

```
public int ResultCode {get;}
```

Property Value

An integer value that indicates if the command completed successfully.

Remarks

The **ResultCode** property is a three-digit numeric code that is used to indicate success or failure. These codes are defined as part of the Hypertext Transfer Protocol standard, with values in the range of 200-299 indicating success. Values in the range of 400-499 and 500-599 indicate failure due to various error conditions. Examples of such failures would be attempting to access a file that does not exist, issuing an unrecognized command or attempting to perform a privileged operation.

See Also

HttpServer.ResultEventArgs Class | SocketTools Namespace

HttpServer.OnStart Event

Occurs when the server starts accepting connections.

[Visual Basic]

```
Public Event OnStart As EventHandler
```

[C#]

```
public event EventHandler OnStart;
```

Remarks

The **OnStart** event occurs after the **Start** method has been called and the server and begins listening for connections from clients. An application can use this event to update the user interface and perform any additional initialization functions that are required by the application

See Also

HttpServer Class | SocketTools Namespace

HttpServer.OnStop Event

Occurs when the server stops accepting connections.

[Visual Basic]

```
Public Event OnStop As EventHandler
```

[C#]

```
public event EventHandler OnStop;
```

Remarks

The **OnStop** event occurs after the **Stop** method has been called and all active client sessions have terminated. An application can use this event to update the user interface and perform any additional cleanup functions that are required by the application. If the server has a large number of active clients, this event may not occur immediately. The **OnDisconnect** event will fire for each client as the server is in the process of shutting down. During the shutdown process, the server is still considered to be active, however it will not accept any further connections. When the **OnStop** event is fired, the server thread has terminated and the listening socket has been closed.

This event will not occur if the server is forcibly stopped using the **Reset** method, or when the **Uninitialize** method is called prior to disposing an instance of the control. Applications that depend on this event should ensure that the server is shutdown gracefully using the **Stop** method prior to terminating the application

See Also

HttpServer Class | SocketTools Namespace

HttpServer.OnTimeout Event

Occurs when the client has exceeded the maximum allowed idle time.

[Visual Basic]
```
Public Event OnTimeout As OnTimeoutEventHandler
```

[C#]
```
public event OnTimeoutEventHandler OnTimeout;
```

Event Data

The event handler receives an argument of type HttpServer.TimeoutEventArgs containing data related to this event. The following **HttpServer.TimeoutEventArgs** properties provide information specific to this event.

Property	Description
ClientId	Gets a value that uniquely identifies the client session.
Elapsed	Gets the amount of time that the client was idle.

Remarks

The **OnTimeout** event occurs after the client has has exceeded the maximum allowed idle time, and immediately before the client is disconnected from the server. This event will never occur during a file transfer or directory listing.

To change the default idle timeout period for all clients, set the **IdleTime** property prior to starting the server. To set the idle timeout period for a specific client, set the **ClientIdle** property in the **OnConnect** event handler.

This event handler will be invoked in the context of the worker thread that is managing the client session, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.TimeoutEventArgs Class

Provides data for the OnTimeout event.

For a list of all members of this type, see HttpServer.TimeoutEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.HttpServer.TimeoutEventArgs

[Visual Basic]
```
Public Class HttpServer.TimeoutEventArgs
    Inherits EventArgs
```

[C#]
```
public class HttpServer.TimeoutEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

HttpServer.TimeoutEventArgs Members | SocketTools Namespace | OnTimeout Event (SocketTools.HttpServer)

HttpServer.TimeoutEventArgs Constructor

Initializes a new instance of the HttpServer.TimeoutEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public HttpServer.TimeoutEventArgs();
```

See Also

HttpServer.TimeoutEventArgs Class | SocketTools Namespace

HttpServer.TimeoutEventArgs Members

HttpServer.TimeoutEventArgs overview

Public Instance Constructors

HttpServer.TimeoutEventArgs Constructor	Initializes a new instance of the HttpServer.TimeoutEventArgs class.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
Elapsed	Gets the amount of time that the client was idle.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpServer.TimeoutEventArgs Class | SocketTools Namespace | OnTimeout Event (SocketTools.HttpServer)

HttpServer.TimeoutEventArgs Properties

The properties of the **HttpServer.TimeoutEventArgs** class are listed below. For a complete list of **HttpServer.TimeoutEventArgs** class members, see the HttpServer.TimeoutEventArgs Members topic.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
Elapsed	Gets the amount of time that the client was idle.

See Also

HttpServer.TimeoutEventArgs Class | SocketTools Namespace | OnTimeout Event (SocketTools.HttpServer)

HttpServer.TimeoutEventArgs.ClientId Property

Gets a value that uniquely identifies the client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which identifies the client session.

Remarks

The **ClientId** property returns a unique integer value that identifies the client session that generated the event.

See Also

HttpServer.TimeoutEventArgs Class | SocketTools Namespace

HttpServer.OnUpload Event

Occurs when the client has successfully uploaded a file to the server.

[Visual Basic]

```
Public Event OnUpload As OnUploadEventHandler
```

[C#]

```
public event OnUploadEventHandler OnUpload;
```

Event Data

The event handler receives an argument of type HttpServer.UploadEventArgs containing data related to this event. The following **HttpServer.UploadEventArgs** properties provide information specific to this event.

Property	Description
ClientId	Gets a value that uniquely identifies the client session.
FileName	Gets a value that specifies the file being uploaded.
FileSize	Gets a value that specifies the size of the file.

Remarks

The **OnUpload** event occurs after the client has successfully uploaded a file to the server using the PUT command. If the file transfer fails or is aborted, this event will not occur.

This event handler will be invoked in the context of the worker thread that is managing the client session, not the thread that created an instance of the class. Because UI components should only be modified by the thread that created them, the event handler should never attempt to update the user interface directly.

See Also

HttpServer Class | SocketTools Namespace

HttpServer.UploadEventArgs Class

Provides data for the OnUpload event.

For a list of all members of this type, see HttpServer.UploadEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.HttpServer.UploadEventArgs

```
[Visual Basic]
Public Class HttpServer.UploadEventArgs
    Inherits EventArgs
```

```
[C#]
public class HttpServer.UploadEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

HttpServer.UploadEventArgs Members | SocketTools Namespace

HttpServer.UploadEventArgs Constructor

Initializes a new instance of the HttpServer.UploadEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public HttpServer.UploadEventArgs();
```

See Also

HttpServer.UploadEventArgs Class | SocketTools Namespace

HttpServer.UploadEventArgs Members

HttpServer.UploadEventArgs overview

Public Instance Constructors

HttpServer.UploadEventArgs Constructor	Initializes a new instance of the HttpServer.UploadEventArgs class.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
FileName	Gets a value that specifies the file being uploaded.
FileSize	Gets a value that specifies the size of the file.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

HttpServer.UploadEventArgs Class | SocketTools Namespace

HttpServer.UploadEventArgs Properties

The properties of the **HttpServer.UploadEventArgs** class are listed below. For a complete list of **HttpServer.UploadEventArgs** class members, see the HttpServer.UploadEventArgs Members topic.

Public Instance Properties

ClientId	Gets a value that uniquely identifies the client session.
FileName	Gets a value that specifies the file being uploaded.
FileSize	Gets a value that specifies the size of the file.

See Also

HttpServer.UploadEventArgs Class | SocketTools Namespace

HttpServer.UploadEventArgs.ClientId Property

Gets a value that uniquely identifies the client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which identifies the client session.

Remarks

The **ClientId** property returns a unique integer value that identifies the client session that generated the event.

See Also

HttpServer.UploadEventArgs Class | SocketTools Namespace

HttpServer.UploadEventArgs.FileName Property

Gets a value that specifies the file being uploaded.

[Visual Basic]

```
Public ReadOnly Property FileName As String
```

[C#]

```
public string FileName {get;}
```

Property Value

A string that specifies the full path to a file on the local system.

See Also

HttpServer.UploadEventArgs Class | SocketTools Namespace

HttpServer.UploadEventArgs.FileSize Property

Gets a value that specifies the size of the file.

[Visual Basic]

```
Public ReadOnly Property FileSize As Long
```

[C#]

```
public long FileSize {get;}
```

Property Value

A long integer value that specifies the size of the file in bytes.

See Also

HttpServer.UploadEventArgs Class | SocketTools Namespace

HttpServer.OnAuthenticateEventHandler Delegate

Represents the method that will handle the OnAuthenticate event.

[Visual Basic]

```
Public Delegate Sub HttpServer.OnAuthenticateEventHandler( _
   ByVal sender As Object, _
   ByVal e As AuthenticateEventArgs _
)
```

[C#]

```
public delegate void HttpServer.OnAuthenticateEventHandler(
      object sender,
      AuthenticateEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An AuthenticateEventArgs that contains the event data.

Remarks

When you create an **OnAuthenticateEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnAuthenticateEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

SocketTools Namespace

HttpServer.OnCommandEventHandler Delegate

Represents the method that will handle the OnCommand event.

[Visual Basic]

```
Public Delegate Sub HttpServer.OnCommandEventHandler( _
   ByVal sender As Object, _
   ByVal e As CommandEventArgs _
)
```

[C#]

```
public delegate void HttpServer.OnCommandEventHandler(
      object sender,
      CommandEventArgs e
   );
```

Parameters

sender
The source of the event.

e
An CommandEventArgs that contains the event data.

Remarks

When you create an **OnCommandEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnCommandEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

SocketTools Namespace

HttpServer.OnConnectEventHandler Delegate

Represents the method that will handle the OnConnect event.

[Visual Basic]

```
Public Delegate Sub HttpServer.OnConnectEventHandler( _
   ByVal sender As Object, _
   ByVal e As ConnectEventArgs _
)
```

[C#]

```
public delegate void HttpServer.OnConnectEventHandler(
      object sender,
      ConnectEventArgs e
   );
```

Parameters

sender
: The source of the event.

e
: An ConnectEventArgs that contains the event data.

Remarks

When you create an **OnConnectEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnConnectEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

SocketTools Namespace

HttpServer.OnDisconnectEventHandler Delegate

Represents the method that will handle the OnDisconnect event.

[Visual Basic]

```
Public Delegate Sub HttpServer.OnDisconnectEventHandler( _
   ByVal sender As Object, _
   ByVal e As DisconnectEventArgs _
)
```

[C#]

```
public delegate void HttpServer.OnDisconnectEventHandler(
      object sender,
      DisconnectEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An DisconnectEventArgs that contains the event data.

Remarks

When you create an **OnDisconnectEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnDisconnectEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

SocketTools Namespace

HttpServer.OnDownloadEventHandler Delegate

Represents the method that will handle the OnDownload event.

[Visual Basic]

```
Public Delegate Sub HttpServer.OnDownloadEventHandler( _
   ByVal sender As Object, _
   ByVal e As DownloadEventArgs _
)
```

[C#]

```
public delegate void HttpServer.OnDownloadEventHandler(
      object sender,
      DownloadEventArgs e
   );
```

Parameters

sender
The source of the event.

e
An DownloadEventArgs that contains the event data.

Remarks

When you create an **OnDownloadEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnDownloadEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

SocketTools Namespace

HttpServer.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub HttpServer.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void HttpServer.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ErrorEventArgs that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

SocketTools Namespace

HttpServer.OnExecuteEventHandler Delegate

Represents the method that will handle the OnExecute event.

[Visual Basic]

```
Public Delegate Sub HttpServer.OnExecuteEventHandler( _
   ByVal sender As Object, _
   ByVal e As ExecuteEventArgs _
)
```

[C#]

```
public delegate void HttpServer.OnExecuteEventHandler(
      object sender,
      ExecuteEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ExecuteEventArgs that contains the event data.

Remarks

When you create an **OnExecuteEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnExecuteEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

SocketTools Namespace

HttpServer.OnResultEventHandler Delegate

Represents the method that will handle the OnResult event.

[Visual Basic]

```
Public Delegate Sub HttpServer.OnResultEventHandler( _
   ByVal sender As Object, _
   ByVal e As ResultEventArgs _
)
```

[C#]

```
public delegate void HttpServer.OnResultEventHandler(
      object sender,
      ResultEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ResultEventArgs that contains the event data.

Remarks

When you create an **OnResultEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnResultEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

SocketTools Namespace

HttpServer.OnTimeoutEventHandler Delegate

Represents the method that will handle the OnTimeout event.

[Visual Basic]

```
Public Delegate Sub HttpServer.OnTimeoutEventHandler( _
   ByVal sender As Object, _
   ByVal e As TimeoutEventArgs _
)
```

[C#]

```
public delegate void HttpServer.OnTimeoutEventHandler(
      object sender,
      TimeoutEventArgs e
   );
```

Parameters

sender
The source of the event.

e
An TimeoutEventArgs that contains the event data.

Remarks

When you create an **OnTimeoutEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnTimeoutEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

SocketTools Namespace

HttpServer.OnUploadEventHandler Delegate

Represents the method that will handle the OnUpload event.

[Visual Basic]

```
Public Delegate Sub HttpServer.OnUploadEventHandler( _
   ByVal sender As Object, _
   ByVal e As UploadEventArgs _
)
```

[C#]

```
public delegate void HttpServer.OnUploadEventHandler(
      object sender,
      UploadEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An UploadEventArgs that contains the event data.

Remarks

When you create an **OnUploadEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnUploadEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

SocketTools Namespace

HttpServer.FileAccess Enumeration

Specifies the file access permissions the HttpServer class supports.

[Visual Basic]

```
Public Enum HttpServer.FileAccess
```

[C#]

```
public enum HttpServer.FileAccess
```

Remarks

File access permissions can restrict the actions that any client can take, regardless of the user permissions assigned to the client session. For example, a client session may have permission to use the PUT command; however, unless the folder that they are attempting to create the file in also has **accessWrite** permission, the PUT command will fail.

For security reasons, when the server is started, regular files only have the **accessRead** permission and directories only have the **accessRead** and **accessList** permissions assigned to them. If you wish to allow clients to upload files to your server, or execute scripts stored in a directory, then you must create a virtual path to a physical directory and assign it the appropriate permissions. In both cases, best practices dictate that the physical directory should be located outside of the root directory of the server.

If you assign the **accessExecute** permission to a virtual directory to allow clients to execute scripts using the GET or POST commands, you should make sure that clients cannot list, create or modify files in that directory. The scripts in that directory must have a registered handler, created using the **RegisterHandler** method. It is not necessary to create a virtual path to a CGI program registered using the **RegisterProgram** method because execute permission for that program is granted by default.

Members

Member Name	Description
accessNone	The virtual path or file has not been assigned any permissions.
accessRead	If the virtual path specifies a file, the client can use the GET command to retrieve the contents of the file and the HEAD command will return information about the file. If the virtual path specifies a directory, the client can use the GET command to retrieve the index file for that directory. If the file or directory does not have this permission, the server will return an error to the client.
accessWrite	If the virtual path specifies a file, the client can modify the contents of the file using the PUT command. If the path specifies a directory, the client can use the PUT command to create a new file or replace an existing file in the directory.
accessExecute	If the virtual path specifies a script, the client can execute the script using either the GET or POST commands. If the path specifies a directory, then all scripts in that directory can be executed.

accessList	If the virtual path specifies a directory, and there is no default index file present, the server will return a list of files in that directory to the client. If this permission is not specified, the server will return an error if the directory does not have a default index file. It is recommended that you do not specify this permission when assigning the **accessExecute** permission to a directory
accessRestricted	Access to the file or directory should be restricted to using the GET command to retrieve documents. This is effectively the same as only specifying **accessRead** as the file access permissions. If this permission is combined with any permission other than **accessRead**, those permissions will be ignored.
accessProtected	Access to the file or directory is protected by a username and password. Clients should only be permitted to access the resource if they provide valid user credentials to the server. If this permission is assigned to a virtual path, the default command handlers will require the client to authenticate itself to permit access to the resource. The server application is responsible for authenticating the session.
accessDefault	This value specifies that the default access permissions should be granted to the file or directory. If the virtual path specifies a file, the client can use the GET command to return the contents. If the path specifies a directory, the client can use the GET command to return the index file or a list of files in the directory. If the server is in restricted mode, it will return an error if a directory does not have an index page.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

SocketTools Namespace

HttpServer.ErrorCode Enumeration

Specifies the error codes returned by the HttpServer class.

[Visual Basic]

```
Public Enum HttpServer.ErrorCode
```

[C#]

```
public enum HttpServer.ErrorCode
```

Remarks

The HttpServer class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This method has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.
errorProxyAuthenticationRequired	Proxy authentication required.

errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.
errorUnsupportedMediaType	Request specified an unsupported media type.

errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.
errorNoMoreMailboxes	No more mailboxes exist on this server.

errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.
errorInvalidThreadHandle	The specified thread handle is no longer valid.

errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been defined.

errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or unrecognized.

errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

HttpServer.FormatType Enumeration

Specifies the logfile formats that the HttpServer class supports.

[Visual Basic]
```
Public Enum HttpServer.FormatType
```

[C#]
```
public enum HttpServer.FormatType
```

Members

Member Name	Description
formatNone	This value specifies that the server should not create or update a log file. This is the default property value.
formatCommon	This value specifies that the log file should use the common log format that records a subset of information in a fixed format. This log format usually only provides information about file transfers.
formatCombined	This value specifies that the server should use the combined log file format. This format is similar to the common format, however it includes the client referrer and user agent. This is the format that most Apache web servers use by default.
formatExtended	This value specifies that the log file should use the standard W3C extended log file format. This is an extensible format that can provide additional information about the client session.

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

SocketTools Namespace

HttpServer.SecurityProtocols Enumeration

Specifies the security protocols that the HttpServer class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum HttpServer.SecurityProtocols
```

[C#]

```
[Flags]
public enum HttpServer.SecurityProtocols
```

Remarks

The HttpServer class uses the **SecurityProtocols** enumeration to specify one or more security protocols to be used when establishing a connection with a remote host. Multiple protocols may be specified if necessary and the actual protocol used will be negotiated with the remote host. It is recommended that most applications use **protocolDefault** when starting a secure server.

Members

Member Name	Description	Value
protocolNone	No security protocol will be used, a secure connection will not be established.	0
protocolSSL2	The SSL 2.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections.	1
protocolSSL3	The SSL 3.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections. In most cases, this protocol is only selected if TLS is not supported by the server.	2
protocolTLS10	The TLS 1.0 protocol has been selected. This version of the protocol is commonly used by older servers and is the only version of TLS supported on Windows platforms prior to Windows 7 SP1 and Windows Server 2008 R2.	4
protocolTLS11	The TLS 1.1 protocol has been selected. This version of TLS is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of the	8

	operating system.	
protocolTLS12	The TLS 1.2 protocol has been selected. This is the default version of the protocol and is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of Windows. It is recommended that you use this version of TLS.	16
protocolSSL	Any version of the Secure Sockets Layer (SSL) protocol should be used. The actual protocol version used will be negotiated with the remote host.	3
protocolTLS	Any version of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with with preference for TLS 1.2.	28
protocolTLS1	Version 1.0, 1.1 or 1.2 of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with preference for TLS 1.2.	28
protocolDefault	The default selection of security protocols will be used when establishing a connection. The TLS 1.2, 1.1 and 1.0 protocols will be negotiated with the host, in that order of preference. This option will always request the latest version of the preferred security protocols and is the recommended value.	16
protocolUnknown	An unknown or unsupported security protocol has been specified. This value indicates an error condition.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

SocketTools Namespace

HttpServer.ServerOptions Enumeration

Specifies the options that the HttpServer class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum HttpServer.ServerOptions
```

[C#]

```
[Flags]
public enum HttpServer.ServerOptions
```

Remarks

The HttpServer class uses the **ServerOptions** enumeration to specify one or more options to be used when establishing a connection with a remote host. Multiple options may be specified if necessary.

Members

Member Name	Description	Value
optionNone	No option specified.	0
optionMultiUser	This option specifies the server should be started in multi-user mode, where users are assigned their own home directories and clients can access documents in those directories by including the username in the request URI. If this option is not specified, then all users will share the server root directory by default. This option does not have any effect on the maximum number of simultaneous client sessions that can be established with the server.	1
optionRestricted	This option specifies the server should be initialized in a restricted mode, limiting certain functionality. The only commands accepted by the server will be the GET and HEAD commands. The server will never return a list of files if the client provides a URL that maps to a local directory and there is no default index page. Clients will not be able to execute CGI programs or scripts, and cannot access files outside of the server root directory or its subdirectories.	2
optionLocalUser	This option specifies the server should perform user authentication using the Windows local account database. This option is useful if the server should accept local usernames, or if the	4

	application does not wish to implement an event handler for user authentication. If this option is not specified, the application is responsible for authenticating all users.	
optionNoIndex	This option specifies the server should not search for a default index page if the client provides a URL that maps to a local directory. By default, the server will search for a file named index.htm, index.html, default.htm, default.html or index.txt in the directory. If a file by one of those names is found, it will return the contents of that file rather than a list of files in the directory.	8
optionReadOnly	This option specifies the server should only allow read-only access to files by default. If this option is enabled, it will change the default permissions granted to authenticated users. Commands that are used to create, delete or modify files on the server will be disabled by default. It is recommended that this option be enabled if the server is publicly accessible over the Internet.	16
optionLockFiles	This option specifies that files should be exclusively locked when a client attempts to upload or download a file. If another client attempts to access the same file, the operation will fail. By default, the server will permit multiple clients to access the same file, although it will still write-lock files that are in the process of being uploaded.	64
optionHiddenFiles	This option specifies that when a client requests a resource, the server should permit access to hidden and system files or subdirectories. By default, the server will not allow access to a hidden or system file, even if the client session has been authenticated. This option is ignored if the server is started in restricted mode.	128
optionSecure	This option specifies that secure connections using SSL and/or TLS should be enabled. This option requires that a valid SSL certificate be installed on the local host. The default port number for secure HTTP connections is 443. If security is enabled, all client	4096

	connections to the server must be secure. Standard and secure connections cannot be shared by the same instance of the server. If your application must support both standard and secure connections, you must create two instances of the server listening on two different ports, one with **optionSecure** enabled and the other without.	
optionSecureFallback	This option specifies the server should permit the use of less secure cipher suites for compatibility with legacy clients. If this option is specified, the server will permit connections using TLS 1.0 and cipher suites that use RC4, MD5 and SHA1.	32768

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

SocketTools Namespace

HttpServer.ServerPriority Enumeration

Specifies the priorities that the HttpServer class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum HttpServer.ServerPriority
```

```
[C#]
[Flags]
public enum HttpServer.ServerPriority
```

Members

Member Name	Description	Value
priorityBackground	This priority significantly reduces the memory, processor and network resource utilization for the server. It is typically used with lightweight services running in the background that are designed for few client connections. Each client thread will be assigned a lower scheduling priority and will be frequently forced to yield execution to other threads.	0
priorityLow	This priority lowers the overall resource utilization for the client session and meters the processor utilization for the client session. Each client thread will be assigned a lower scheduling priority and will occasionally be forced to yield execution to other threads.	1
priorityNormal	The default priority which balances resource and processor utilization. It is recommended that most applications use this priority.	2
priorityHigh	This priority increases the overall resource utilization for each client session and their threads will be given higher scheduling priority. It is not recommended that this priority be used on a system with a single processor.	3
priorityCritical	This priority can significantly increase processor, memory and network utilization. Each client thread will be given higher scheduling priority and will be more responsive to network events. It is not recommended that this priority be used on a system with a single	4

	processor.	
priorityInvalid	An invalid transfer priority which indicates an error condition.	-1
priorityDefault	The default server priority. This is the same as specifying **priorityNormal**.	2
priorityLowest	The lowest valid server priority. This is the same as specifying **priorityBackground**.	0
priorityHighest	The highest valid server priority. This is the same as specifying **priorityCritical**.	4

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

SocketTools Namespace

HttpServer.TraceOptions Enumeration

Specifies the logging options that the HttpServer class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum HttpServer.TraceOptions
```

[C#]

```
[Flags]
public enum HttpServer.TraceOptions
```

Remarks

The HttpServer class uses the **TraceOptions** enumeration to specify what kind of debugging information is written to the trace logfile. These options are only meaningful when trace logging is enabled by setting the **Trace** property to **true**.

Members

Member Name	Description	Value
traceDefault	The default trace logging option. This is the same as specifying the **traceInfo** option.	0
traceInfo	All network function calls are written to the trace file. This is the default value.	0
traceError	Only those network function calls which fail are recorded in the trace file.	1
traceWarning	Only those network function calls which fail, or return values which indicate a warning, are recorded in the trace file.	2
traceHexDump	All network function calls are written to the trace file, plus all the data that is sent or received is displayed, in both ASCII and hexadecimal format.	4
traceProcess	All function calls in the current process are logged, rather than only those functions in the current thread. This option is useful for multithreaded applications that are using worker threads.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

SocketTools Namespace

HttpServer.UserAccess Enumeration

Specifies the user access permissions the HttpServer class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum HttpServer.UserAccess
```

[C#]

```
[Flags]
public enum HttpServer.UserAccess
```

Remarks

When a client establishes a connection to the server, it is granted a default set of user access permissions based on the initial configuration of the server. By default, the client is granted all permissions, which means the client may use any valid HTTP command. If the server is started in restricted mode, then the client is only granted permission to read files. This means that restricted mode clients cannot obtain directory listings of files, nor can they create files or execute CGI programs. The user access permissions define the types of HTTP commands that the client is permitted to use. However, server options and individual permissions on specific files and directories can further limit what actions the client can take.

If you assign the **accesRestricted** permission to a client, the server will impose significant limitations on the client. This permission provides a high level of security, ensuring that the client cannot access any other documents outside of the server root directory; however, it also prevents the client from executing scripts or submitting data. If the website depends on server-side scripts and the use of CGI programs, assigning this permission may effectively disable use of the site for that client session.

Members

Member Name	Description	Value
accessNone	The client has not been assigned any permissions.	0
accessAll	The client has been assigned all available permissions. This enables the client to issue all valid commands, including any external script handlers that have been registered for use with the server instance.	65535
accessRead	The client can download files and retrieve other resources using the GET command. This permission also allows the client to obtain information about a specific resource using the HEAD command. The resource that the client is attempting to retrieve must also have read permission, otherwise the command will fail.	1
accessWrite	The client can modify existing files or create new files using the PUT	2

	command. The directory where the client is attempting to create or modify the file must also have write permission, otherwise the command will fail. This permission is not granted by default to clients if the server is started in restricted mode. This permission is ignored if the server is started in read-only mode.	
accessExecute	The client can execute scripts and CGI programs. If this permission is not granted to the client, it will be unable to use the GET, HEAD or POST commands if the resource is a program or script registered with the server. This permission is not granted by default to clients if the server is started in restricted mode.	4
accessList	If the client issues a GET command and the resource specifies a directory, this permission allows the server to return a list of files to client if a default index file cannot be found. If this permission is not granted to the client, the directory must contain a default index file, otherwise the server will return an error. This permission is ignored if the server is started in restricted mode.	8
accessRestricted	The client is restricted to accessing documents using the GET and HEAD commands, and those documents must be located in the root directory for the virtual host or in a subdirectory. The client cannot execute scripts, submit data to the server using the POST command or upload files using the PUT command.	1048576
accessDefault	This value specifies that the default permissions should be granted to the client session. If the server is in restricted mode, the client will only be able to use the GET and HEAD commands to retrieve documents. If the server is not in restricted mode, the client can use all valid HTTP commands. This is the recommended access permissions for most clients.	8388608

Requirements

Namespace: SocketTools

Assembly: SocketTools.HttpServer (in SocketTools.HttpServer.dll)

See Also

SocketTools Namespace

IcmpClient Class

Implements the Internet Control Message Protocol.

For a list of all members of this type, see IcmpClient Members.

System.Object
  SocketTools.IcmpClient

[Visual Basic]
```
Public Class IcmpClient
    Implements IDisposable
```

[C#]
```
public class IcmpClient : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The Internet Control Message Protocol (ICMP) is commonly used to determine if a remote host is reachable and how packets of data are routed to that system. Users are most familiar with this protocol as it is implemented in the ping and tracert command line utilities. The ping command is used to check if a system is reachable and the amount of time that it takes for a packet of data to make a round trip from the local system, to the remote host and then back again. The tracert command is used to trace the route that a packet of data takes from the local system to the remote host, and can be used to identify potential problems with overall throughput and latency. The IcmpClient class can be used to build in this type of functionality in your own applications, giving you the ability to send and receive ICMP echo datagrams in order to perform your own analysis.

Requirements

Namespace: SocketTools

Assembly: SocketTools.IcmpClient (in SocketTools.IcmpClient.dll)

See Also

IcmpClient Members | SocketTools Namespace

IcmpClient Members

IcmpClient overview

Public Static (Shared) Fields

icmpPacketSize	A constant value which specifies the default packet size.
icmpTimeout	A constant value which specifies the default timeout period in milliseconds.
icmpTimeToLive	A constant value which specifies the default time-to-live value.

Public Instance Constructors

IcmpClient Constructor	Initializes a new instance of the IcmpClient class.

Public Instance Properties

AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
Blocking	Gets and sets a value which indicates if the client is in blocking mode.
Handle	Gets a value that specifies the client handle allocated for the current session.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
Interval	Gets and sets the interval in milliseconds between echo packets.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets and sets the local Internet address that the client will be bound to.
LocalName	Gets a value which specifies the host name for the local system.
Options	Gets and sets a value which specifies one or more client options.
PacketSize	Gets and sets the size of an echo datagram.

RecvCount	Gets the number of echo reply datagrams received by the local host.
SendCount	Gets the number of echo datagrams sent by the local host.
Status	Gets a value which specifies the current status of the client.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in milliseconds.
TimeToLive	Gets and sets the default time-to-live value for echo datagrams.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
TripAverage	Gets the average packet trip time in milliseconds.
TripMaximum	Gets the maximum packet trip time in milliseconds.
TripMinimum	Gets the minimum packet trip time in milliseconds.
Version	Gets a value which returns the current version of the IcmpClient class library.

Public Instance Methods

AttachThread	Attach an instance of the class to the current thread
Cancel	Cancel the current blocking client operation.
Dispose	Overloaded. Releases all resources used by IcmpClient.
Echo	Overloaded. Send an echo datagram to the specified host.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the IcmpClient

	class.
Reset	Reset the internal state of the object, resetting all properties to their default values.
ToString (inherited from Object)	Returns a String that represents the current Object.
TraceRoute	Overloaded. Send a series of echo datagrams to trace the route taken from the local system to the remote host
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.

Public Instance Events

OnCancel	Occurs when a blocking client operation is canceled.
OnEcho	Occurs when an echo datagram is sent to the remote host.
OnError	Occurs when an client operation fails.
OnReply	Occurs when an echo reply datagram is received by the local host.
OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
OnTrace	Occurs when an echo datagram is forwarded to an intermediate host.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the IcmpClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient Constructor

Initializes a new instance of the IcmpClient class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public IcmpClient();
```

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient Properties

The properties of the **IcmpClient** class are listed below. For a complete list of **IcmpClient** class members, see the IcmpClient Members topic.

Public Instance Properties

AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
Blocking	Gets and sets a value which indicates if the client is in blocking mode.
Handle	Gets a value that specifies the client handle allocated for the current session.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
Interval	Gets and sets the interval in milliseconds between echo packets.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets and sets the local Internet address that the client will be bound to.
LocalName	Gets a value which specifies the host name for the local system.
Options	Gets and sets a value which specifies one or more client options.
PacketSize	Gets and sets the size of an echo datagram.
RecvCount	Gets the number of echo reply datagrams received by the local host.
SendCount	Gets the number of echo datagrams sent by the local host.
Status	Gets a value which specifies the current status of the client.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method

	calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in milliseconds.
TimeToLive	Gets and sets the default time-to-live value for echo datagrams.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
TripAverage	Gets the average packet trip time in milliseconds.
TripMaximum	Gets the maximum packet trip time in milliseconds.
TripMinimum	Gets the minimum packet trip time in milliseconds.
Version	Gets a value which returns the current version of the IcmpClient class library.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.AutoResolve Property

Gets and sets a value that determines if host names and addresses are automatically resolved.

[Visual Basic]

```
Public Property AutoResolve As Boolean
```

[C#]

```
public bool AutoResolve {get; set;}
```

Property Value

Returns **true** if host names are automatically resolved to Internet addresses. The default value is **false**.

Remarks

Setting the **AutoResolve** property determines if the class automatically resolves host names and addresses specified by the **HostName** and **HostAddress** properties. If set to **true**, setting the **HostName** property will cause the class to automatically determine the corresponding IP address and update the **HostAddress** property accordingly. Likewise, setting the **HostAddress** property will cause the class to determine the host name and update the **HostName** property. Setting this property to **false** prevents the class from resolving host names until a connection attempt is made.

It is important to note that setting the **HostName** or **HostAddress** property may cause the current thread to block, sometimes for several seconds, until the name or address is resolved. To prevent this behavior, set this property value to **false**.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.Blocking Property

Gets and sets a value which indicates if the client is in blocking mode.

[Visual Basic]

```
Public Property Blocking As Boolean
```

[C#]

```
public bool Blocking {get; set;}
```

Property Value

Returns **true** if the client is in blocking mode; otherwise it returns **false**. The default value is **true**.

Remarks

Setting the **Blocking** property determines if client operations complete synchronously or asynchronously. If set to **true**, then each client operation (such as sending or receiving data) will return when the operation has completed or timed-out. If set to **false**, client operations will return immediately. If the operation would result in the client blocking (such as attempting to read data when no data has been sent by the remote host), an error is generated.

It is important to note that certain events, such as **OnDisconnect, OnEcho** and **OnReply** are only fired if the client is in non-blocking mode.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.Handle Property

Gets a value that specifies the client handle allocated for the current session.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer which represents a client handle. If there is no active connection, a value of -1 is returned.

Remarks

The **Handle** property specifies the numeric descriptor of the current client session.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.HostAddress Property

Gets and sets a value which specifies the Internet address used to establish a connection.

[Visual Basic]

```
Public Property HostAddress As String
```

[C#]

```
public string HostAddress {get; set;}
```

Property Value

A string which specifies an Internet address using dotted notation.

Remarks

The **HostAddress** property can be used to set the Internet address for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the address is assigned to a valid host name, the **HostName** property will be updated with that value.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.HostName Property

Gets and sets a value which specifies the host name used to establish a connection.

[Visual Basic]

```
Public Property HostName As String
```

[C#]

```
public string HostName {get; set;}
```

Property Value

A string which specifies a host name.

Remarks

The **HostName** property can be used to set the host name for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the name can be resolved to a valid Internet address, the **HostAddress** property will be updated with that value.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.Interval Property

Gets and sets the interval in milliseconds between echo packets.

[Visual Basic]

```
Public Property Interval As Integer
```

[C#]

```
public int Interval {get; set;}
```

Property Value

An integer value which specifies an interval in milliseconds.

Remarks

The **Interval** property determines the interval at which echo datagrams are automatically sent to the remote host. If the interval is set to zero, no datagrams are automatically sent.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.IsBlocked Property

Gets a value which indicates if the current thread is performing a blocking client operation.

[Visual Basic]

```
Public ReadOnly Property IsBlocked As Boolean
```

[C#]

```
public bool IsBlocked {get;}
```

Property Value

Returns **true** if the current thread is blocking, otherwise returns **false**.

Remarks

The **IsBlocked** property returns **true** if the current thread is blocked performing an operation. Because the Windows Sockets API only permits one blocking operation per thread of execution, this property should be checked before starting any blocking operation in response to an event.

If the **IsBlocked** property returns **false**, this means there are no blocking operations on the current thread at that time. However, this does not guarantee that the next client operation will not fail. An application should always check the return value from a client operation and check the value of the **LastError** property if an error occurs.

Note that this property will return **true** if there is any blocking operation being performed by the current thread, regardless of whether this specific instance of the class is responsible for the blocking operation or not.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.IsInitialized Property

Gets a value which indicates if the current instance of the class has been initialized successfully.

[Visual Basic]

```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]

```
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the class instance has been initialized; otherwise returns **false**.

Remarks

The **IsInitialized** property is used to determine if the current instance of the class has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the class may not be able to initialize itself.

The most common reasons that a class instance may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required networking libraries or not being able to access the system registry.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public IcmpClient.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.LocalAddress Property

Gets and sets the local Internet address that the client will be bound to.

[Visual Basic]

```
Public Property LocalAddress As String
```

[C#]

```
public string LocalAddress {get; set;}
```

Property Value

A string which specifies an Internet address in dotted notation.

Remarks

The **LocalAddress** property is used to specify the local Internet address that the client will be bound to when a connection is established with a remote host. By default this property is not assigned a value, which specifies that the client should be bound to any appropriate network interface on the local system.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.LocalName Property

Gets a value which specifies the host name for the local system.

[Visual Basic]

```
Public ReadOnly Property LocalName As String
```

[C#]

```
public string LocalName {get;}
```

Property Value

A string which specifies the local host name.

Remarks

The **LocalName** property returns the fully-qualified host name assigned to the local system. If the system has not been configured with an Internet domain name, then this property will return the NetBIOS name assigned to the local system.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.Options Property

Gets and sets a value which specifies one or more client options.

[Visual Basic]
```
Public Property Options As IcmpOptions
```

[C#]
```
public IcmpClient.IcmpOptions Options {get; set;}
```

Property Value

Returns one or more IcmpOptions enumeration flags which specify the options for the client. The default value for this property is **clientOptionNone**.

Remarks

The **Options** property specifies one or more default options options which are used when establishing a connection using the **Connect** method.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.PacketSize Property

Gets and sets the size of an echo datagram.

[Visual Basic]

```
Public Property PacketSize As Integer
```

[C#]

```
public int PacketSize {get; set;}
```

Property Value

An integer which specifies the size of an echo datagram in bytes.

Remarks

The **PacketSize** property determines the size of an ICMP echo datagram. The default packet size is 32 bytes. The minimum packet size is 1 byte and the maximum packet size is 65,535 bytes. Specifying a packet size outside of this range will result in an error. Note that packet sizes over 512 bytes may not be supported by your local networking hardware or intermediate routers.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.RecvCount Property

Gets the number of echo reply datagrams received by the local host.

[Visual Basic]

```
Public ReadOnly Property RecvCount As Integer
```

[C#]

```
public int RecvCount {get;}
```

Property Value

An integer value which specifies the number of echo reply datagrams.

Remarks

This value is automatically reset whenever a new remote host is specified.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.SendCount Property

Gets the number of echo datagrams sent by the local host.

[Visual Basic]

```
Public ReadOnly Property SendCount As Integer
```

[C#]

```
public int SendCount {get;}
```

Property Value

An integer value which specifies the number of echo datagrams sent.

Remarks

This value is automatically reset whenever a new remote host is specified.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.Status Property

Gets a value which specifies the current status of the client.

[Visual Basic]

```
Public ReadOnly Property Status As IcmpStatus
```

[C#]

```
public IcmpClient.IcmpStatus Status {get;}
```

Property Value

A IcmpStatus enumeration value which specifies the current client status.

Remarks

The **Status** property returns the current status of the client. This property can be used to check on blocking connections to determine if the client is interacting with the remote host before taking some action.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.ThreadModel Property

Gets and sets a value which specifies the threading model for the class instance.

[Visual Basic]
```
Public Property ThreadModel As ThreadingModel
```

[C#]
```
public IcmpClient.ThreadingModel ThreadModel {get; set;}
```

Property Value

Returns one or more ThreadingModel enumeration value which specifies the threading model for the client. The default value for this property is **modelSingleThread**.

Remarks

The **ThreadModel** property specifies the threading model that is used by the class instance when a connection is established. The default value for this property is **modelSingleThread**, which specifies that only the thread that established the connection should be permitted to invoke methods. It is important to note that this threading model does not limit the application to a single thread of execution. When a session is established using the **Connect** method, that session is attached to the thread that created it. From that point on, until the session is terminated, only the owner may invoke methods in that instance of the class. The ownership of the class instance may be transferred from one thread to another using the **AttachThread** method.

Setting this property to **modelFreeThread** disables certain internal safety checks that are performed by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads. The application must ensure that no two threads will attempt to invoke a blocking method at the same time. In other words, if one thread invokes a method, the application must ensure that another thread will not attempt to invoke any other method at the same time using the same instance of the class.

See Also

IcmpClient Class | SocketTools Namespace | AttachThread Method | ThreadingModel Enumeration | ThreadModel Attribute

IcmpClient.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]

```
Public Property ThrowError As Boolean
```

[C#]

```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.Timeout Property

Gets and sets a value which specifies a timeout period in milliseconds.

[Visual Basic]

```
Public Property Timeout As Integer
```

[C#]

```
public int Timeout {get; set;}
```

Property Value

An integer value which specifies a timeout period in milliseconds.

Remarks

Setting the **Timeout** property specifies the number of milliseconds until a blocking operation fails and returns an error.

The timeout period is only used when the client is in blocking mode. Although this property can be changed when the client is in non-blocking mode, the value will be ignored until the client is returned to blocking mode.

For most applications it is recommended the timeout period be set to 3000 milliseconds (3 seconds).

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.TimeToLive Property

Gets and sets the default time-to-live value for echo datagrams.

[Visual Basic]

```
Public Property TimeToLive As Integer
```

[C#]

```
public int TimeToLive {get; set;}
```

Remarks

The time-to-live (TTL) value is specified in the IP header of a datagram, and is used to control the number of routers that the datagram is passed through. Each router that handles the datagram decrements the TTL value by one. When it drops to zero, a datagram is returned to the sender, specifying that the TTL has been exceeded.

Setting this property changes the default TTL value for all subsequent ICMP datagrams sent by the local host, with the default value being 255. Reading this property returns the value of the TTL field in the IP header of the last echo reply datagram received.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.Trace Property

Gets and sets a value which indicates if network function logging is enabled.

[Visual Basic]

```
Public Property Trace As Boolean
```

[C#]

```
public bool Trace {get; set;}
```

Property Value

Returns **true** if network function tracing is enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Trace** property is used to enable (or disable) the tracing of network function calls. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the SocketTools networking classes and is enabled or disabled for an entire process. This means that once trace logging is enabled for a given component, all of the function calls made by the process using any of the SocketTools classes will be logged. For example, if you have an application using both the File Transfer Protocol and Post Office Protocol classes, and you set the **Trace** property to **true**, function calls made by both classes will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all classes used by the process.

If trace logging is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the logfile is fairly large. Since logfiles can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

When redistributing your application, make sure that you include the **SocketTools10.TraceLog.dll** module with your installation. If this library is not present, then no trace output will be generated and the value of the **Trace** property will be ignored. Only those function calls made by the SocketTools networking classes will be logged. Calls made directly to the Windows Sockets API, or calls made by other classes, will not be logged.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.TraceFile Property

Gets and sets a value which specifies the name of the logfile.

[Visual Basic]
```
Public Property TraceFile As String
```

[C#]
```
public string TraceFile {get; set;}
```

Property Value

A string which specifies the name of the file.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named **SocketTools.log** is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFile** property should be set to the same value for each instance. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
 MyApp INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0 MyApp WRN:
connect(46, 192.0.0.1:1234, 16) returned -1 [10035] MyApp ERR: accept(46,
NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the control is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Note that if the specified file cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.TraceFlags Property

Gets and sets a value which specifies the client function tracing flags.

[Visual Basic]

```
Public Property TraceFlags As TraceOptions
```

[C#]

```
public IcmpClient.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration which specifies the amount of detail written to the trace logfile.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Because network function tracing is enabled per-process, the trace flags are shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFlags** property should be set to the same value for each instance. Changing the trace flags for any one instance of the class will affect the logging performed for all SocketTools classes used by the application.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being written and the error **errorOperationWouldBlock** occurs, a warning is generated because the application simply needs to attempt to write the data at a later time.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.TripAverage Property

Gets the average packet trip time in milliseconds.

[Visual Basic]

```
Public ReadOnly Property TripAverage As Integer
```

[C#]

```
public int TripAverage {get;}
```

Property Value

An integer value which specifies the average trip time in milliseconds.

Remarks

The **TripAverage** property returns the average number of milliseconds for an ICMP echo reply datagram to be returned from the remote host.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.TripMaximum Property

Gets the maximum packet trip time in milliseconds.

[Visual Basic]

```
Public ReadOnly Property TripMaximum As Integer
```

[C#]

```
public int TripMaximum {get;}
```

Property Value

An integer value which specifies the maximum trip time in milliseconds.

Remarks

The **TripMaximum** property returns the maximum number of milliseconds for an ICMP echo reply datagram to be returned from the remote host.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.TripMinimum Property

Gets the minimum packet trip time in milliseconds.

[Visual Basic]

```
Public ReadOnly Property TripMinimum As Integer
```

[C#]

```
public int TripMinimum {get;}
```

Property Value

An integer value which specifies the minimum trip time in milliseconds.

Remarks

The **TripMinimum** property returns the minimum number of milliseconds for an ICMP echo reply datagram to be returned from the remote host.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.Version Property

Gets a value which returns the current version of the IcmpClient class library.

[Visual Basic]

```
Public ReadOnly Property Version As String
```

[C#]

```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the IcmpClient class library. This value can be used by an application for validation and debugging purposes.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient Methods

The methods of the **IcmpClient** class are listed below. For a complete list of **IcmpClient** class members, see the IcmpClient Members topic.

Public Instance Methods

AttachThread	Attach an instance of the class to the current thread
Cancel	Cancel the current blocking client operation.
Dispose	Overloaded. Releases all resources used by IcmpClient.
Echo	Overloaded. Send an echo datagram to the specified host.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the IcmpClient class.
Reset	Reset the internal state of the object, resetting all properties to their default values.
ToString (inherited from Object)	Returns a String that represents the current Object.
TraceRoute	Overloaded. Send a series of echo datagrams to trace the route taken from the local system to the remote host
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the IcmpClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.AttachThread Method

Attach an instance of the class to the current thread

[Visual Basic]
```
Public Function AttachThread() As Boolean
```

[C#]
```
public bool AttachThread();
```

Return Value

A boolean value which specifies if the client could be attached to the current thread. If this method returns **false**, the client could not be attached to the thread and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

When an instance of the class is created it is associated with the current thread that created it. Normally, if another thread attempts to perform an operation using that instance, an error is returned. This is used to ensure that other threads cannot interfere with an operation being performed by the owner thread. In some cases, it may be desirable for one thread in an application to create an instance of the class, establish a connection and then pass that instance to another worker thread. The **AttachThread** method can be used to change the ownership of the class instance to the new worker thread.

This method should be called by the new thread immediately after it has been created, and if the new thread does not release the handle itself, the ownership of the handle should be restored by the original thread. Under no circumstances should **AttachThread** be used to forcibly destroy an instance of a class allocated by another thread while a blocking operation is in progress. To cancel a blocking operation, use the **Cancel** method and then delete the class instance after the blocking function exits and control is returned to the current thread.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.Cancel Method

Cancel the current blocking client operation.

[Visual Basic]

```
Public Sub Cancel()
```

[C#]

```
public void Cancel();
```

Remarks

When the **Cancel** method is called, the blocking client operation will not immediately fail. An internal flag is set which causes the blocking operation to exit with an error. This means that the application cannot cancel an operation and immediately perform some other blocking function. Instead it must allow the calling stack to unwind, returning back to the blocking operation before making any further function calls.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.Echo Method

Send an echo datagram to the default host.

Overload List

Send an echo datagram to the default host.

public bool Echo();

Send an echo datagram to the specified host.

public bool Echo(string);

Send an echo datagram to the specified host.

public bool Echo(string,int);

Send an echo datagram to the specified host.

public bool Echo(string,int,int);

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.Echo Method ()

Send an echo datagram to the default host.

[Visual Basic]

```
Overloads Public Function Echo() As Boolean
```

[C#]

```
public bool Echo();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Echo** method sends an ICMP echo datagram to the specified host, providing a simplified interface for pinging a remote system. If the method returns a value of **true**, then a reply was received for the echo datagram that was sent. This would typically indicate that the client can establish a reliable connection to the server. A return value of **false** indicates that there was no response to the echo datagrams. The remote host may not exist or may not be available.

The value returned by the **TripAverage** property provides information about the latency between the two hosts. Higher average time values would indicate greater latency and reduced throughput between the systems.

The failure for a host to respond to an ICMP echo datagram may not indicate a problem with the remote system. In some cases, a router between the local and remote host may be malfunctioning or discarding the datagrams. Systems can also be configured to specifically ignore ICMP echo datagrams and not respond to them; this is often a security measure to prevent certain kinds of denial-of-service attacks.

The value of the **HostName** property specifies the remote host that the echo datagram will be sent to. The **Timeout** property specifies the timeout period in milliseconds. The **TimeToLive** property specifies the time-to-live value.

See Also

IcmpClient Class | SocketTools Namespace | IcmpClient.Echo Overload List

IcmpClient.Echo Method (String)

Send an echo datagram to the specified host.

[Visual Basic]

```
Overloads Public Function Echo( _
   ByVal hostName As String _
) As Boolean
```

[C#]

```
public bool Echo(
   string hostName
);
```

Parameters

hostName

A string which specifies the host name or IP address which the echo datagram will be sent to.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Echo** method sends an ICMP echo datagram to the specified host, providing a simplified interface for pinging a remote system. If the method returns a value of **true**, then a reply was received for the echo datagram that was sent. This would typically indicate that the client can establish a reliable connection to the server. A return value of **false** indicates that there was no response to the echo datagrams. The remote host may not exist or may not be available.

The value returned by the **TripAverage** property provides information about the latency between the two hosts. Higher average time values would indicate greater latency and reduced throughput between the systems.

The failure for a host to respond to an ICMP echo datagram may not indicate a problem with the remote system. In some cases, a router between the local and remote host may be malfunctioning or discarding the datagrams. Systems can also be configured to specifically ignore ICMP echo datagrams and not respond to them; this is often a security measure to prevent certain kinds of denial-of-service attacks.

The **Timeout** property specifies the timeout period in milliseconds. The **TimeToLive** property specifies the time-to-live value.

See Also

IcmpClient Class | SocketTools Namespace | IcmpClient.Echo Overload List

IcmpClient.Echo Method (String, Int32)

Send an echo datagram to the specified host.

[Visual Basic]

```
Overloads Public Function Echo( _
   ByVal hostName As String, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Echo(
   string hostName,
   int timeout
);
```

Parameters

hostName
A string which specifies the host name or IP address which the echo datagram will be sent to.

timeout
An integer value which specifies the number of milliseconds until a blocking operation fails and the method returns to the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Echo** method sends an ICMP echo datagram to the specified host, providing a simplified interface for pinging a remote system. If the method returns a value of **true**, then a reply was received for the echo datagram that was sent. This would typically indicate that the client can establish a reliable connection to the server. A return value of **false** indicates that there was no response to the echo datagrams. The remote host may not exist or may not be available.

The value returned by the **TripAverage** property provides information about the latency between the two hosts. Higher average time values would indicate greater latency and reduced throughput between the systems.

The failure for a host to respond to an ICMP echo datagram may not indicate a problem with the remote system. In some cases, a router between the local and remote host may be malfunctioning or discarding the datagrams. Systems can also be configured to specifically ignore ICMP echo datagrams and not respond to them; this is often a security measure to prevent certain kinds of denial-of-service attacks.

The **TimeToLive** property specifies the time-to-live value.

See Also

IcmpClient Class | SocketTools Namespace | IcmpClient.Echo Overload List

IcmpClient.Echo Method (String, Int32, Int32)

Send an echo datagram to the specified host.

[Visual Basic]

```
Overloads Public Function Echo( _
   ByVal hostName As String, _
   ByVal timeout As Integer, _
   ByVal timeToLive As Integer _
) As Boolean
```

[C#]

```
public bool Echo(
   string hostName,
   int timeout,
   int timeToLive
);
```

Parameters

hostName
A string which specifies the host name or IP address which the echo datagram will be sent to.

timeout
An integer value which specifies the number of milliseconds until a blocking operation fails and the method returns to the caller.

timeToLive
An integer value which specifies the time-to-live (TTL) value for the echo datagram. This determines the maximum number of times that a packet will be forwarded from one system to another while enroute to its destination. The minimum time-to-live value is 1, the maximum is 255.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Echo** method sends an ICMP echo datagram to the specified host, providing a simplified interface for pinging a remote system. If the method returns a value of **true**, then a reply was received for the echo datagram that was sent. This would typically indicate that the client can establish a reliable connection to the server. A return value of **false** indicates that there was no response to the echo datagrams. The remote host may not exist or may not be available.

The value returned by the **TripAverage** property provides information about the latency between the two hosts. Higher average time values would indicate greater latency and reduced throughput between the systems.

The failure for a host to respond to an ICMP echo datagram may not indicate a problem with the remote system. In some cases, a router between the local and remote host may be malfunctioning or discarding the datagrams. Systems can also be configured to specifically ignore ICMP echo datagrams and not respond to them; this is often a security measure to prevent certain kinds of denial-of-service attacks.

See Also

IcmpClient Class | SocketTools Namespace | IcmpClient.Echo Overload List

IcmpClient.Dispose Method

Releases all resources used by IcmpClient.

Overload List

Releases all resources used by IcmpClient.

public void Dispose();

Releases the unmanaged resources allocated by the IcmpClient class and optionally releases the managed resources.

protected virtual void Dispose(bool);

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.Dispose Method ()

Releases all resources used by IcmpClient.

[Visual Basic]

```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]

```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

IcmpClient Class | SocketTools Namespace | IcmpClient.Dispose Overload List

IcmpClient.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the IcmpClient class and optionally releases the managed resources.

[Visual Basic]

```
Overridable Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]

```
protected virtual void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **IcmpClient** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

IcmpClient Class | SocketTools Namespace | IcmpClient.Dispose Overload List

IcmpClient.Finalize Method

Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.

[Visual Basic]

```
Overrides Protected Sub Finalize()
```

[C#]

```
protected override void Finalize();
```

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.Initialize Method

Initialize an instance of the IcmpClient class.

Overload List

Initialize an instance of the IcmpClient class.

public bool Initialize();

Initialize an instance of the IcmpClient class.

public bool Initialize(string);

See Also

IcmpClient Class | SocketTools Namespace | Uninitialize Method

IcmpClient.Initialize Method ()

Initialize an instance of the IcmpClient class.

[Visual Basic]
```
Overloads Public Function Initialize() As Boolean
```

[C#]
```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the IcmpClient class, loading the networking library and allocating resources for the current thread. Typically it is not necessary to explicitly call this method because the instance of the class is initialized by the class constructor. However, if the **Uninitialize** method is called, the class must be re-initialized before any other methods are called.

See Also

IcmpClient Class | SocketTools Namespace | IcmpClient.Initialize Overload List | Uninitialize Method

IcmpClient.Initialize Method (String)

Initialize an instance of the IcmpClient class.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the IcmpClient class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the IcmpClient class.

Example

The following example shows how to use the Initialize method to initialize an instance of the class. This example assumes that the license key string has been defined in code.

```
SocketTools.IcmpClient icmpClient = new SocketTools.IcmpClient();

if (icmpClient.Initialize(strLicenseKey) == false)
{
    MessageBox.Show(icmpClient.LastErrorString, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    return;
}
```

```
Dim icmpClient As New SocketTools.IcmpClient

If icmpClient.Initialize(strLicenseKey) = False Then
    MsgBox(icmpClient.LastErrorString, vbIconExclamation)
    Exit Sub
End If
```

See Also

IcmpClient Class | SocketTools Namespace | IcmpClient.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

IcmpClient.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. If a socket has been allocated, it will be released and any active connections will be terminated. All properties will be reset to their default values.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.TraceRoute Method

Send a series of echo datagrams to trace the route taken from the local system to the remote host

Overload List

Send a series of echo datagrams to trace the route taken from the local system to the remote host

public int TraceRoute();

Send a series of echo datagrams to trace the route taken from the local system to the remote host

public int TraceRoute(string);

Send a series of echo datagrams to trace the route taken from the local system to the remote host

public int TraceRoute(string,int);

Send a series of echo datagrams to trace the route taken from the local system to the remote host

public int TraceRoute(string,int,int);

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.TraceRoute Method ()

Send a series of echo datagrams to trace the route taken from the local system to the remote host

[Visual Basic]

```
Overloads Public Function TraceRoute() As Integer
```

[C#]

```
public int TraceRoute();
```

Return Value

The method returns the total number of hops from the local system to the remote host. If the method fails, it will return a value of -1.

Remarks

The **TraceRoute** method sends a series of ICMP echo datagrams to the specified host, adjusting the time-to-live value to determine the intermediate hosts that route the packet. This method will always block the current thread until the trace completes.

The **OnTrace** event will fire for each intermediate host along the route.

The value of the **HostName** property specifies the remote host that the echo datagram will be sent to. The **Timeout** property specifies the timeout period in milliseconds.

See Also

IcmpClient Class | SocketTools Namespace | IcmpClient.TraceRoute Overload List

IcmpClient.TraceRoute Method (String)

Send a series of echo datagrams to trace the route taken from the local system to the remote host

[Visual Basic]

```
Overloads Public Function TraceRoute( _
   ByVal hostName As String _
) As Integer
```

[C#]

```
public int TraceRoute(
   string hostName
);
```

Parameters

hostName

A string which specifies the host name or IP address which the echo datagram will be sent to.

Return Value

The method returns the total number of hops from the local system to the remote host. If the method fails, it will return a value of -1.

Remarks

The **TraceRoute** method sends a series of echo datagrams to the specified host, adjusting the time-to-live value to determine the intermediate hosts that route the packet. This method will always block the current thread until the trace completes.

The **OnTrace** event will fire for each intermediate host along the route.

The **Timeout** property specifies the timeout period in milliseconds.

See Also

IcmpClient Class | SocketTools Namespace | IcmpClient.TraceRoute Overload List

IcmpClient.TraceRoute Method (String, Int32)

Send a series of echo datagrams to trace the route taken from the local system to the remote host

[Visual Basic]

```
Overloads Public Function TraceRoute( _
   ByVal hostName As String, _
   ByVal maxHops As Integer _
) As Integer
```

[C#]

```
public int TraceRoute(
   string hostName,
   int maxHops
);
```

Parameters

hostName

A string which specifies the host name or IP address which the echo datagram will be sent to.

maxHops

An integer value which specifies the maximum number of routers the echo datagram will be forwarded through (the number of hops) to the remote host. The minimum value is 1 and the maximum value is 255. It is recommended that most applications specify a value of at least 30.

Return Value

The method returns the total number of hops from the local system to the remote host. If the method fails, it will return a value of -1.

Remarks

The **TraceRoute** method sends a series of echo datagrams to the specified host, adjusting the time-to-live value to determine the intermediate hosts that route the packet. This method will always block the current thread until the trace completes.

The **OnTrace** event will fire for each intermediate host along the route.

The **Timeout** property specifies the timeout period in milliseconds.

See Also

IcmpClient Class | SocketTools Namespace | IcmpClient.TraceRoute Overload List

IcmpClient.TraceRoute Method (String, Int32, Int32)

Send a series of echo datagrams to trace the route taken from the local system to the remote host

[Visual Basic]

```
Overloads Public Function TraceRoute( _
   ByVal hostName As String, _
   ByVal maxHops As Integer, _
   ByVal timeout As Integer _
) As Integer
```

[C#]

```
public int TraceRoute(
   string hostName,
   int maxHops,
   int timeout
);
```

Parameters

hostName

A string which specifies the host name or IP address which the echo datagram will be sent to.

maxHops

An integer value which specifies the maximum number of routers the echo datagram will be forwarded through (the number of hops) to the remote host. The minimum value is 1 and the maximum value is 255. It is recommended that most applications specify a value of at least 30.

timeout

An integer value which specifies the number of milliseconds until a blocking operation fails and the method returns to the caller.

Return Value

The method returns the total number of hops from the local system to the remote host. If the method fails, it will return a value of -1.

Remarks

The **TraceRoute** method sends a series of echo datagrams to the specified host, adjusting the time-to-live value to determine the intermediate hosts that route the packet. This method will always block the current thread until the trace completes.

The **OnTrace** event will fire for each intermediate host along the route.

See Also

IcmpClient Class | SocketTools Namespace | IcmpClient.TraceRoute Overload List

IcmpClient.Uninitialize Method

Uninitialize the class library and release any resources allocated for the current thread.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method terminates any active connection, releases resources allocated for the current thread and unloads the networking library. After this method has been called, no further client operations may be performed until the class instance has been re-initialized.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the class is no longer needed.

See Also

IcmpClient Class | SocketTools Namespace | Initialize Method

IcmpClient Events

The events of the **IcmpClient** class are listed below. For a complete list of **IcmpClient** class members, see the IcmpClient Members topic.

Public Instance Events

⚡ OnCancel	Occurs when a blocking client operation is canceled.
⚡ OnEcho	Occurs when an echo datagram is sent to the remote host.
⚡ OnError	Occurs when an client operation fails.
⚡ OnReply	Occurs when an echo reply datagram is received by the local host.
⚡ OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
⚡ OnTrace	Occurs when an echo datagram is forwarded to an intermediate host.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.OnCancel Event

Occurs when a blocking client operation is canceled.

[Visual Basic]

```
Public Event OnCancel As EventHandler
```

[C#]

```
public event EventHandler OnCancel;
```

Remarks

The **OnCancel** event is generated when a blocking client operation, such as sending or receiving data, is canceled with the **Cancel** method. To assist in determining which operation was canceled, check the value of the **Status** property.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.OnEcho Event

Occurs when an echo datagram is sent to the remote host.

[Visual Basic]
```
Public Event OnEcho As OnEchoEventHandler
```

[C#]
```
public event OnEchoEventHandler OnEcho;
```

Event Data

The event handler receives an argument of type IcmpClient.EchoEventArgs containing data related to this event. The following **IcmpClient.EchoEventArgs** properties provide information specific to this event.

Property	Description
HostName	Gets the remote host name or IP address.
PacketSize	Gets the size of the echo datagram in bytes.
SequenceId	Gets the packet sequence number.

Remarks

The **OnEcho** event is generated for non-blocking sockets when an echo datagram is sent to the remote host. This event is only generated when the **Blocking** property is set to **false**.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.EchoEventArgs Class

Provides data for the OnEcho event.

For a list of all members of this type, see IcmpClient.EchoEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.IcmpClient.EchoEventArgs

[Visual Basic]

```
Public Class IcmpClient.EchoEventArgs
    Inherits EventArgs
```

[C#]

```
public class IcmpClient.EchoEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

EchoEventArgs provides information about the echo datagram sent to the remote host.

An OnEcho event occurs when an echo datagram is sent to a remote host.

Requirements

Namespace: SocketTools

Assembly: SocketTools.IcmpClient (in SocketTools.IcmpClient.dll)

See Also

IcmpClient.EchoEventArgs Members | SocketTools Namespace

IcmpClient.EchoEventArgs Members

IcmpClient.EchoEventArgs overview

Public Instance Constructors

IcmpClient.EchoEventArgs Constructor	Initializes a new instance of the IcmpClient.EchoEventArgs class.

Public Instance Properties

HostName	Gets the remote host name or IP address.
PacketSize	Gets the size of the echo datagram in bytes.
SequenceId	Gets the packet sequence number.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

IcmpClient.EchoEventArgs Class | SocketTools Namespace

IcmpClient.EchoEventArgs Constructor

Initializes a new instance of the IcmpClient.EchoEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public IcmpClient.EchoEventArgs();
```

See Also

IcmpClient.EchoEventArgs Class | SocketTools Namespace

IcmpClient.EchoEventArgs Properties

The properties of the **IcmpClient.EchoEventArgs** class are listed below. For a complete list of **IcmpClient.EchoEventArgs** class members, see the IcmpClient.EchoEventArgs Members topic.

Public Instance Properties

HostName	Gets the remote host name or IP address.
PacketSize	Gets the size of the echo datagram in bytes.
SequenceId	Gets the packet sequence number.

See Also

IcmpClient.EchoEventArgs Class | SocketTools Namespace

IcmpClient.EchoEventArgs.HostName Property

Gets the remote host name or IP address.

[Visual Basic]

```
Public ReadOnly Property HostName As String
```

[C#]

```
public string HostName {get;}
```

Property Value

A string which specifies the remote host name or IP address.

See Also

IcmpClient.EchoEventArgs Class | SocketTools Namespace

IcmpClient.EchoEventArgs.PacketSize Property

Gets the size of the echo datagram in bytes.

[Visual Basic]

```
Public ReadOnly Property PacketSize As Integer
```

[C#]

```
public int PacketSize {get;}
```

Property Value

An integer value which specifies the size of the echo datagram in bytes.

See Also

IcmpClient.EchoEventArgs Class | SocketTools Namespace

IcmpClient.EchoEventArgs.SequenceId Property

Gets the packet sequence number.

[Visual Basic]
```
Public ReadOnly Property SequenceId As Integer
```

[C#]
```
public int SequenceId {get;}
```

Property Value

An integer which specifies the packet sequence number.

Remarks

The **SequenceId** value is used to uniquely identify each echo datagram that is sent to the remote host. This value will increase for each datagram that is sent until the remote host address is changed. Once a new remote host is specified, the sequence number is reset.

See Also

IcmpClient.EchoEventArgs Class | SocketTools Namespace

IcmpClient.OnError Event

Occurs when an client operation fails.

[Visual Basic]

```
Public Event OnError As OnErrorEventHandler
```

[C#]

```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type IcmpClient.ErrorEventArgs containing data related to this event. The following **IcmpClient.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a client operation fails.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see IcmpClient.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.IcmpClient.ErrorEventArgs

[Visual Basic]
```
Public Class IcmpClient.ErrorEventArgs
    Inherits EventArgs
```

[C#]
```
public class IcmpClient.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.IcmpClient (in SocketTools.IcmpClient.dll)

See Also

IcmpClient.ErrorEventArgs Members | SocketTools Namespace

IcmpClient.ErrorEventArgs Members

IcmpClient.ErrorEventArgs overview

Public Instance Constructors

IcmpClient.ErrorEventArgs Constructor	Initializes a new instance of the IcmpClient.ErrorEventArgs class.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

IcmpClient.ErrorEventArgs Class | SocketTools Namespace

IcmpClient.ErrorEventArgs Constructor

Initializes a new instance of the IcmpClient.ErrorEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public IcmpClient.ErrorEventArgs();
```

See Also

IcmpClient.ErrorEventArgs Class | SocketTools Namespace

IcmpClient.ErrorEventArgs Properties

The properties of the **IcmpClient.ErrorEventArgs** class are listed below. For a complete list of **IcmpClient.ErrorEventArgs** class members, see the IcmpClient.ErrorEventArgs Members topic.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

IcmpClient.ErrorEventArgs Class | SocketTools Namespace

IcmpClient.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

IcmpClient.ErrorEventArgs Class | SocketTools Namespace | Error Property

IcmpClient.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public IcmpClient.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

IcmpClient.ErrorEventArgs Class | SocketTools Namespace | Description Property

IcmpClient.OnReply Event

Occurs when an echo reply datagram is received by the local host.

[Visual Basic]

```
Public Event OnReply As OnReplyEventHandler
```

[C#]

```
public event OnReplyEventHandler OnReply;
```

Event Data

The event handler receives an argument of type IcmpClient.ReplyEventArgs containing data related to this event. The following **IcmpClient.ReplyEventArgs** properties provide information specific to this event.

Property	Description
HostName	Gets the remote host name or IP address.
PacketSize	Gets the size of the echo datagram in bytes.
SequenceId	Gets the packet sequence number.
TripTime	Gets the time in milliseconds since the packet was sent to the remote host.

Remarks

This **OnReply** event is fired when a echo reply datagram is received from the remote system. Note that there is no guarantee that packets will be returned in the same sequence order they were sent or that they will be returned at all. This event is only generated when the **Blocking** property is set to **false**.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.ReplyEventArgs Class

Provides data for the OnReply event.

For a list of all members of this type, see IcmpClient.ReplyEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.IcmpClient.ReplyEventArgs

[Visual Basic]
```
Public Class IcmpClient.ReplyEventArgs
    Inherits EventArgs
```

[C#]
```
public class IcmpClient.ReplyEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ReplyEventArgs provides information about the echo reply datagram received from the remote host.

An OnReply event occurs when an echo reply datagram is received from a remote host.

Requirements

Namespace: SocketTools

Assembly: SocketTools.IcmpClient (in SocketTools.IcmpClient.dll)

See Also

IcmpClient.ReplyEventArgs Members | SocketTools Namespace

IcmpClient.ReplyEventArgs Members

IcmpClient.ReplyEventArgs overview

Public Instance Constructors

IcmpClient.ReplyEventArgs Constructor	Initializes a new instance of the IcmpClient.ReplyEventArgs class.

Public Instance Properties

HostName	Gets the remote host name or IP address.
PacketSize	Gets the size of the echo datagram in bytes.
SequenceId	Gets the packet sequence number.
TripTime	Gets the time in milliseconds since the packet was sent to the remote host.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

IcmpClient.ReplyEventArgs Class | SocketTools Namespace

IcmpClient.ReplyEventArgs Constructor

Initializes a new instance of the IcmpClient.ReplyEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public IcmpClient.ReplyEventArgs();
```

See Also

IcmpClient.ReplyEventArgs Class | SocketTools Namespace

IcmpClient.ReplyEventArgs Properties

The properties of the **IcmpClient.ReplyEventArgs** class are listed below. For a complete list of **IcmpClient.ReplyEventArgs** class members, see the IcmpClient.ReplyEventArgs Members topic.

Public Instance Properties

HostName	Gets the remote host name or IP address.
PacketSize	Gets the size of the echo datagram in bytes.
SequenceId	Gets the packet sequence number.
TripTime	Gets the time in milliseconds since the packet was sent to the remote host.

See Also

IcmpClient.ReplyEventArgs Class | SocketTools Namespace

IcmpClient.ReplyEventArgs.HostName Property

Gets the remote host name or IP address.

[Visual Basic]

```
Public ReadOnly Property HostName As String
```

[C#]

```
public string HostName {get;}
```

Property Value

A string which specifies the remote host name or IP address.

See Also

IcmpClient.ReplyEventArgs Class | SocketTools Namespace

IcmpClient.ReplyEventArgs.PacketSize Property

Gets the size of the echo datagram in bytes.

[Visual Basic]

```
Public ReadOnly Property PacketSize As Integer
```

[C#]

```
public int PacketSize {get;}
```

Property Value

An integer value which specifies the size of the echo datagram in bytes.

See Also

IcmpClient.ReplyEventArgs Class | SocketTools Namespace

IcmpClient.ReplyEventArgs.SequenceId Property

Gets the packet sequence number.

[Visual Basic]

```
Public ReadOnly Property SequenceId As Integer
```

[C#]

```
public int SequenceId {get;}
```

Property Value

An integer which specifies the packet sequence number.

Remarks

The **SequenceId** value is used to uniquely identify each echo datagram that is received from the remote host. This value will increase for each datagram that is received until the remote host address is changed. Once a new remote host is specified, the sequence number is reset.

See Also

IcmpClient.ReplyEventArgs Class | SocketTools Namespace

IcmpClient.ReplyEventArgs.TripTime Property

Gets the time in milliseconds since the packet was sent to the remote host.

[Visual Basic]

```
Public ReadOnly Property TripTime As Integer
```

[C#]

```
public int TripTime {get;}
```

Property Value

An integer which specifies the number of milliseconds that have elapsed since the packet was sent to the remote host.

Remarks

This value can be used to measure the latency between the local system and remote host.

See Also

IcmpClient.ReplyEventArgs Class | SocketTools Namespace

IcmpClient.OnTimeout Event

Occurs when a blocking operation fails to complete before the timeout period elapses.

[Visual Basic]

```
Public Event OnTimeout As EventHandler
```

[C#]

```
public event EventHandler OnTimeout;
```

Remarks

The **OnTimeout** event occurs when a blocking operation, such as sending or receiving data on the client, fails to complete before the specified timeout period elapses. The timeout period for a blocking operation can be adjusted by setting the **Timeout** property.

This event is only generated if the client is in blocking mode.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.OnTrace Event

Occurs when an echo datagram is forwarded to an intermediate host.

[Visual Basic]
```
Public Event OnTrace As OnTraceEventHandler
```

[C#]
```
public event OnTraceEventHandler OnTrace;
```

Event Data

The event handler receives an argument of type IcmpClient.TraceEventArgs containing data related to this event. The following **IcmpClient.TraceEventArgs** properties provide information specific to this event.

Property	Description
HostName	Gets the remote host name or IP address.
Replies	Gets the number of echo reply datagrams received from the remote host.
TraceHop	Gets a value which specifies the distance from the local system to the specified host.
TripAverage	Gets the average packet trip time in milliseconds.
TripMaximum	Gets the maximum packet trip time in milliseconds.
TripMinimum	Gets the minimum packet trip time in milliseconds.

Remarks

The **OnTrace** event is generated when the **TraceRoute** method is called. This event will fire for each intermediate host in the route from the local system and the remote host.

See Also

IcmpClient Class | SocketTools Namespace

IcmpClient.TraceEventArgs Class

Provides data for the OnTrace event.

For a list of all members of this type, see IcmpClient.TraceEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.IcmpClient.TraceEventArgs

[Visual Basic]
```
Public Class IcmpClient.TraceEventArgs
    Inherits EventArgs
```

[C#]
```
public class IcmpClient.TraceEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

TraceEventArgs provides information about an intermediate host as an echo datagram is forwarded from one system to another.

An OnTrace event occurs when the **TraceRoute** method is called.

Requirements

Namespace: SocketTools

Assembly: SocketTools.IcmpClient (in SocketTools.IcmpClient.dll)

See Also

IcmpClient.TraceEventArgs Members | SocketTools Namespace

IcmpClient.TraceEventArgs Members

IcmpClient.TraceEventArgs overview

Public Instance Constructors

IcmpClient.TraceEventArgs Constructor	Initializes a new instance of the IcmpClient.TraceEventArgs class.

Public Instance Properties

HostName	Gets the remote host name or IP address.
Replies	Gets the number of echo reply datagrams received from the remote host.
TraceHop	Gets a value which specifies the distance from the local system to the specified host.
TripAverage	Gets the average packet trip time in milliseconds.
TripMaximum	Gets the maximum packet trip time in milliseconds.
TripMinimum	Gets the minimum packet trip time in milliseconds.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

IcmpClient.TraceEventArgs Class | SocketTools Namespace

IcmpClient.TraceEventArgs Constructor

Initializes a new instance of the IcmpClient.TraceEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public IcmpClient.TraceEventArgs();
```

See Also

IcmpClient.TraceEventArgs Class | SocketTools Namespace

IcmpClient.TraceEventArgs Properties

The properties of the **IcmpClient.TraceEventArgs** class are listed below. For a complete list of **IcmpClient.TraceEventArgs** class members, see the IcmpClient.TraceEventArgs Members topic.

Public Instance Properties

HostName	Gets the remote host name or IP address.
Replies	Gets the number of echo reply datagrams received from the remote host.
TraceHop	Gets a value which specifies the distance from the local system to the specified host.
TripAverage	Gets the average packet trip time in milliseconds.
TripMaximum	Gets the maximum packet trip time in milliseconds.
TripMinimum	Gets the minimum packet trip time in milliseconds.

See Also

IcmpClient.TraceEventArgs Class | SocketTools Namespace

IcmpClient.TraceEventArgs.HostName Property

Gets the remote host name or IP address.

[Visual Basic]

```
Public ReadOnly Property HostName As String
```

[C#]

```
public string HostName {get;}
```

Property Value

A string which specifies the remote host name or IP address.

See Also

IcmpClient.TraceEventArgs Class | SocketTools Namespace

IcmpClient.TraceEventArgs.Replies Property

Gets the number of echo reply datagrams received from the remote host.

[Visual Basic]

```
Public ReadOnly Property Replies As Integer
```

[C#]

```
public int Replies {get;}
```

Property Value

An integer value which specifies the number of echo reply datagrams received from the remote host.

See Also

IcmpClient.TraceEventArgs Class | SocketTools Namespace

IcmpClient.TraceEventArgs.TraceHop Property

Gets a value which specifies the distance from the local system to the specified host.

[Visual Basic]

```
Public ReadOnly Property TraceHop As Integer
```

[C#]

```
public int TraceHop {get;}
```

Property Value

An integer which specifies distance from the local system to the specified host.

Remarks

This value represents the number of times that the packet was forwarded through a router, also known as the number of "hops" to the remote host. With a traceroute, this value will start at one and increment by one for each intermediate host until the destination is reached or the operation times out.

See Also

IcmpClient.TraceEventArgs Class | SocketTools Namespace

IcmpClient.TraceEventArgs.TripAverage Property

Gets the average packet trip time in milliseconds.

[Visual Basic]

```
Public ReadOnly Property TripAverage As Integer
```

[C#]

```
public int TripAverage {get;}
```

Property Value

An integer value which specifies the average trip time in milliseconds.

See Also

IcmpClient.TraceEventArgs Class | SocketTools Namespace

IcmpClient.TraceEventArgs.TripMaximum Property

Gets the maximum packet trip time in milliseconds.

[Visual Basic]

```
Public ReadOnly Property TripMaximum As Integer
```

[C#]

```
public int TripMaximum {get;}
```

Property Value

An integer value which specifies the maximum trip time in milliseconds.

See Also

IcmpClient.TraceEventArgs Class | SocketTools Namespace

IcmpClient.TraceEventArgs.TripMinimum Property

Gets the minimum packet trip time in milliseconds.

[Visual Basic]

```
Public ReadOnly Property TripMinimum As Integer
```

[C#]

```
public int TripMinimum {get;}
```

Property Value

An integer value which specifies the minimum trip time in milliseconds.

See Also

IcmpClient.TraceEventArgs Class | SocketTools Namespace

IcmpClient.ErrorCode Enumeration

Specifies the error codes returned by the IcmpClient class.

[Visual Basic]

```
Public Enum IcmpClient.ErrorCode
```

[C#]

```
public enum IcmpClient.ErrorCode
```

Remarks

The IcmpClient class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.
errorProxyAuthenticationRequired	Proxy authentication required.

errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.
errorUnsupportedMediaType	Request specified an unsupported media type.

errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.
errorNoMoreMailboxes	No more mailboxes exist on this server.

errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.
errorInvalidThreadHandle	The specified thread handle is no longer valid.

errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been defined.

errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or unrecognized.

errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.

Requirements

Namespace: SocketTools

Assembly: SocketTools.IcmpClient (in SocketTools.IcmpClient.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

IcmpClient.IcmpOptions Enumeration

Specifies the options that the IcmpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum IcmpClient.IcmpOptions
```

```
[C#]
[Flags]
public enum IcmpClient.IcmpOptions
```

Remarks

The IcmpClient class uses the **IcmpOptions** enumeration to specify one or more options to be used when establishing a connection with a remote host. Multiple options may be specified if necessary.

There are currently no additional options for IcmpClient class. This enumeration is provided for future expansion.

Members

Member Name	Description	Value
optionNone	No option specified.	0
optionDefault	The default connection option. This is the same as specifying **optionNone**.	0
optionFreeThread	This option specifies that class methods may be called from any thread, and not only the thread that established the connection. Using this option disables certain internal safety checks that are made by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads.	32768

Requirements

Namespace: SocketTools

Assembly: SocketTools.IcmpClient (in SocketTools.IcmpClient.dll)

See Also

SocketTools Namespace

IcmpClient.IcmpStatus Enumeration

Specifies the status values that may be returned by the IcmpClient class.

[Visual Basic]
```
Public Enum IcmpClient.IcmpStatus
```

[C#]
```
public enum IcmpClient.IcmpStatus
```

Remarks

The IcmpClient class uses the **IcmpStatus** enumeration to identify the current status of the client.

Members

Member Name	Description
statusUnused	A client session has not been created. Attempts to perform any network operations, such as sending or receiving data, will generate an error.
statusIdle	A client session has been created, but is not currently in use. A blocking socket operation can be executed at this point.
statusEcho	An ICMP datagram is being sent to the remote host.
statusReply	An ICMP datagram is being received from the remote host.

Requirements

Namespace: SocketTools

Assembly: SocketTools.IcmpClient (in SocketTools.IcmpClient.dll)

See Also

SocketTools Namespace

IcmpClient.TraceOptions Enumeration

Specifies the logging options that the IcmpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum IcmpClient.TraceOptions
```

[C#]

```
[Flags]
public enum IcmpClient.TraceOptions
```

Remarks

The IcmpClient class uses the **TraceOptions** enumeration to specify what kind of debugging information is written to the trace logfile. These options are only meaningful when trace logging is enabled by setting the **Trace** property to **true**.

Members

Member Name	Description	Value
traceDefault	The default trace logging option. This is the same as specifying the **traceInfo** option.	0
traceInfo	All network function calls are written to the trace file. This is the default value.	0
traceError	Only those network function calls which fail are recorded in the trace file.	1
traceWarning	Only those network function calls which fail, or return values which indicate a warning, are recorded in the trace file.	2
traceHexDump	All network function calls are written to the trace file, plus all the data that is sent or received is displayed, in both ASCII and hexadecimal format.	4
traceProcess	All function calls in the current process are logged, rather than only those functions in the current thread. This option is useful for multithreaded applications that are using worker threads.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.IcmpClient (in SocketTools.IcmpClient.dll)

See Also

SocketTools Namespace

IcmpClient.OnEchoEventHandler Delegate

Represents the method that will handle the OnEcho event.

[Visual Basic]

```
Public Delegate Sub IcmpClient.OnEchoEventHandler( _
   ByVal sender As Object, _
   ByVal e As EchoEventArgs _
)
```

[C#]

```
public delegate void IcmpClient.OnEchoEventHandler(
      object sender,
      EchoEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An EchoEventArgs that contains the event data.

Remarks

When you create an **OnEchoEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnEchoEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.IcmpClient (in SocketTools.IcmpClient.dll)

See Also

SocketTools Namespace

IcmpClient.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub IcmpClient.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void IcmpClient.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ErrorEventArgs that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.IcmpClient (in SocketTools.IcmpClient.dll)

See Also

SocketTools Namespace

IcmpClient.OnReplyEventHandler Delegate

Represents the method that will handle the OnReply event.

[Visual Basic]

```
Public Delegate Sub IcmpClient.OnReplyEventHandler( _
   ByVal sender As Object, _
   ByVal e As ReplyEventArgs _
)
```

[C#]

```
public delegate void IcmpClient.OnReplyEventHandler(
      object sender,
      ReplyEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ReplyEventArgs that contains the event data.

Remarks

When you create an **OnReplyEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnReplyEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.IcmpClient (in SocketTools.IcmpClient.dll)

See Also

SocketTools Namespace

IcmpClient.OnTraceEventHandler Delegate

Represents the method that will handle the OnTrace event.

[Visual Basic]

```
Public Delegate Sub IcmpClient.OnTraceEventHandler( _
   ByVal sender As Object, _
   ByVal e As TraceEventArgs _
)
```

[C#]

```
public delegate void IcmpClient.OnTraceEventHandler(
      object sender,
      TraceEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An TraceEventArgs that contains the event data.

Remarks

When you create an **OnTraceEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnTraceEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.IcmpClient (in SocketTools.IcmpClient.dll)

See Also

SocketTools Namespace

IcmpClient.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see IcmpClient.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.IcmpClient.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class IcmpClient.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class IcmpClient.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the IcmpClient class.

Example

```
<Assembly: SocketTools.IcmpClient.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.IcmpClient.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.IcmpClient (in SocketTools.IcmpClient.dll)

See Also

IcmpClient.RuntimeLicenseAttribute Members | SocketTools Namespace

IcmpClient.RuntimeLicenseAttribute Members

IcmpClient.RuntimeLicenseAttribute overview

Public Instance Constructors

IcmpClient.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

IcmpClient.RuntimeLicenseAttribute Class | SocketTools Namespace

IcmpClient.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public IcmpClient.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the IcmpClient class.

See Also

IcmpClient.RuntimeLicenseAttribute Class | SocketTools Namespace

IcmpClient.RuntimeLicenseAttribute Properties

The properties of the **IcmpClient.RuntimeLicenseAttribute** class are listed below. For a complete list of **IcmpClient.RuntimeLicenseAttribute** class members, see the IcmpClient.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

IcmpClient.RuntimeLicenseAttribute Class | SocketTools Namespace

IcmpClient.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

IcmpClient.RuntimeLicenseAttribute Class | SocketTools Namespace

IcmpClientException Class

The exception that is thrown when a client error occurs.

For a list of all members of this type, see IcmpClientException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.IcmpClientException

[Visual Basic]
```
Public Class IcmpClientException
    Inherits ApplicationException
```

[C#]
```
public class IcmpClientException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A IcmpClientException is thrown by the IcmpClient class when an error occurs.

The default constructor for the IcmpClientException class sets the **ErrorCode** property to the last client error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.IcmpClient (in SocketTools.IcmpClient.dll)

See Also

IcmpClientException Members | SocketTools Namespace

IcmpClientException Members

IcmpClientException overview

Public Instance Constructors

IcmpClientException	Overloaded. Initializes a new instance of the IcmpClientException class.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

IcmpClientException Class | SocketTools Namespace

IcmpClientException Constructor

Initializes a new instance of the IcmpClientException class with the last network error code.

Overload List

Initializes a new instance of the IcmpClientException class with the last network error code.

public IcmpClientException();

Initializes a new instance of the IcmpClientException class with a specified error number.

public IcmpClientException(int);

Initializes a new instance of the IcmpClientException class with a specified error message.

public IcmpClientException(string);

Initializes a new instance of the IcmpClientException class with a specified error message and a reference to the inner exception that is the cause of this exception.

public IcmpClientException(string,Exception);

See Also

IcmpClientException Class | SocketTools Namespace

IcmpClientException Constructor ()

Initializes a new instance of the IcmpClientException class with the last network error code.

[Visual Basic]

```
Overloads Public Sub New()
```

[C#]

```
public IcmpClientException();
```

Remarks

The ctor constructor sets the **ErrorCode** property to the last client error that occurred. For more information about the errors that may occur, refer to the IcmpClient.ErrorCode enumeration.

See Also

IcmpClientException Class | SocketTools Namespace | IcmpClientException Constructor Overload List

IcmpClientException Constructor (String)

Initializes a new instance of the IcmpClientException class with a specified error message.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]

```
public IcmpClientException(
   string message
);
```

Parameters

message

The error message that explains the reason for the exception.

Remarks

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

IcmpClientException Class | SocketTools Namespace | IcmpClientException Constructor Overload List

IcmpClientException Constructor (String, Exception)

Initializes a new instance of the IcmpClientException class with a specified error message and a reference to the inner exception that is the cause of this exception.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal innerException As Exception _
)
```

[C#]

```
public IcmpClientException(
   string message,
   Exception innerException
);
```

Parameters

message

The error message that explains the reason for the exception.

innerException

The exception that is the cause of the current exception. If the ***innerException*** parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

IcmpClientException Class | SocketTools Namespace | IcmpClientException Constructor Overload List

IcmpClientException Constructor (Int32)

Initializes a new instance of the IcmpClientException class with a specified error number.

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

```
[C#]
public IcmpClientException(
   int code
);
```

Parameters

code

An integer value which specifies an error code.

Remarks

This constructor sets the **ErrorCode** property to the specified error code. For more information about the errors that may occur, refer to the IcmpClient.ErrorCode enumeration.

See Also

IcmpClientException Class | SocketTools Namespace | IcmpClientException Constructor Overload List

IcmpClientException Properties

The properties of the **IcmpClientException** class are listed below. For a complete list of **IcmpClientException** class members, see the IcmpClientException Members topic.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

IcmpClientException Class | SocketTools Namespace

IcmpClientException.ErrorCode Property

Gets a value which specifies the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property ErrorCode As ErrorCode
```

[C#]

```
public IcmpClient.ErrorCode ErrorCode {get;}
```

Property Value

Returns a IcmpClient.ErrorCode enumeration value which specifies the error code.

Remarks

The **ErrorCode** property returns the error code that specifies the cause of the exception.

The default constructor for the IcmpClientException class sets the error code to the last network error that occurred. For more information about the errors that may occur, refer to the IcmpClient.ErrorCode enumeration.

See Also

IcmpClientException Class | SocketTools Namespace

IcmpClientException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]

```
Overrides Public ReadOnly Property Message As String
```

[C#]

```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

IcmpClientException Class | SocketTools Namespace

IcmpClientException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception. This value is the same as the values returned by the Windows Sockets API. For more information about socket error codes, see the Windows Socket Version 2 API error code documentation in MSDN.

See Also

IcmpClientException Class | SocketTools Namespace

IcmpClientException Methods

The methods of the **IcmpClientException** class are listed below. For a complete list of **IcmpClientException** class members, see the IcmpClientException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

IcmpClientException Class | SocketTools Namespace

IcmpClientException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]

```
Overrides Public Function ToString() As String
```

[C#]

```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

IcmpClientException Class | SocketTools Namespace

ImapClient Class

Implements the Internet Message Access Protocol.

For a list of all members of this type, see ImapClient Members.

System.Object
  SocketTools.ImapClient

[Visual Basic]
```
Public Class ImapClient
    Implements IDisposable
```

[C#]
```
public class ImapClient : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The Internet Message Access Protocol (IMAP) is an application protocol which is used to access a user's email messages which are stored on a mail server. However, unlike the Post Office Protocol (POP) where messages are downloaded and processed on the local system, the messages on an IMAP server are retained on the server and processed remotely. This is ideal for users who need access to a centralized store of messages or have limited bandwidth. For example, traveling salesmen who have notebook computers or mobile users on a wireless network would be ideal candidates for using IMAP.

The ImapClient class implements the current standard for this protocol, and provides functions to retrieve messages, or just certain parts of a message, create and manage mailboxes, search for specific messages based on certain criteria and so on.

This class supports secure connections using the standard TLS protocols.

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

ImapClient Members | SocketTools Namespace

ImapClient Members

ImapClient overview

Public Static (Shared) Fields

imapPortDefault	A constant value which specifies the default port number.
imapPortSecure	A constant value which specifies the default port number for a secure connection.
imapTimeout	A constant value which specifies the default timeout period.

Public Instance Constructors

ImapClient Constructor	Initializes a new instance of the ImapClient class.

Public Instance Fields

Mailbox	Gets the names of the available mailboxes for the current user.
MessagePart	Gets the contents of the specified message part.

Public Instance Properties

Authentication	Gets and sets the method used to authenticate the user.
AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
BearerToken	Gets and sets the bearer token used with OAuth 2.0 authentication.
Blocking	Gets and sets a value which indicates if the client is in blocking mode.
CertificateExpires	Get a value that specifies the date that the security certificate expires.
CertificateIssued	Get a value that specifies the date that the security certificate was issued.
CertificateIssuer	Get a value that provides information about the organization that issued the certificate.
CertificateName	Gets and sets a value that specifies the name of the client certificate.
CertificatePassword	Gets and sets the password associated with the client certificate.
CertificateStatus	Gets a value which indicates the status of the security certificate returned by the remote host.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.

CertificateSubject	Gets a value that provides information about the organization that the server certificate was issued to.
CertificateUser	Gets and sets the user that owns the client certificate.
CipherStrength	Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.
Delimiter	Gets the hierarchical path delimiter used for the current mailbox.
Handle	Gets a value that specifies the client handle allocated for the current session.
HashStrength	Gets a value which specifies the length of the message digest that was selected for a secure connection.
HeaderField	Gets and sets the current header field name.
HeaderValue	Gets the value of the current header field.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsConnected	Gets a value which indicates if a connection to the remote host has been established.
IsIdle	Gets a value which indicates if the client is idle and the current mailbox is being monitored for status changes.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsReadable	Gets a value which indicates if there is data available to be read from the socket connection to the server.
IsWritable	Gets a value which indicates if data can be written to the client without blocking.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets the local Internet address that the client is bound to.
Localize	Gets a value that specifies if the date and time are localized.

LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
Mailboxes	Gets the number of mailboxes available on the server.
MailboxFlags	Gets one or more flags which identify characteristics of the current mailbox.
MailboxMask	Gets and sets the current mailbox wildcard mask.
MailboxName	Gets and sets the name of the current mailbox.
MailboxPath	Gets and sets the current mailbox reference path.
MailboxSize	Gets the size of the current mailbox.
MailboxUID	Gets the unique identifier for the current mailbox.
Message	Gets and sets the current message number.
MessageCount	Gets the number of messages available in the current mailbox.
MessageFlags	Gets and sets one or more flags for the current message.
MessageParts	Gets the number of message parts in the current message.
MessageSize	Gets the size of the current message in bytes.
MessageUID	Gets the UID for the current message.
NewMessages	Gets the number of new messages available in the current mailbox.
Options	Gets and sets a value which specifies one or more client options.
Password	Gets and sets the password used to authenticate the client.
ReadOnly	Gets a value which specifies if the current mailbox can be modified.
RecentMessages	Gets the number of messages which have recently arrived in the mailbox.
RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.
ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.
Secure	Gets and sets a value which specifies if a secure connection is established.

SecureCipher	Gets a value that specifies the encryption algorithm used for a secure connection.
SecureHash	Gets a value that specifies the message digest algorithm used for a secure connection.
SecureKeyExchange	Gets a value that specifies the key exchange algorithm used for a secure connection.
SecureProtocol	Gets and sets a value which specifies the protocol used for a secure connection.
Status	Gets a value which specifies the current status of the client.
Subscribed	Gets a value that specifies if the user has subscribed to the currently selected mailbox.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
TimeZone	Gets and sets the current timezone offset in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
UserName	Gets and sets the username used to authenticate the client.
Version	Gets a value which returns the current version of the ImapClient class library.

Public Instance Methods

AttachThread	Attach an instance of the class to the current thread
Cancel	Cancel the current blocking client operation.
CheckMailbox	Create a checkpoint for the currently selected mailbox.
CloseMessage	Closes the current message.
Command	Overloaded. Send a custom command to the mail server.
Connect	Overloaded. Establish a connection with a remote host.

CopyMessage	Copy a message from the current mailbox to another mailbox.
CreateMailbox	Creates a new mailbox on the server.
CreateMessage	Overloaded. Create a new message.
DeleteMailbox	Overloaded. Deletes a mailbox from the server.
DeleteMessage	Overloaded. Flags a message for deletion from the current mailbox.
Disconnect	Terminate the connection with a remote host.
Dispose	Overloaded. Releases all resources used by ImapClient.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
ExamineMailbox	Selects the specified mailbox for read-only access.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetHeader	Overloaded. Returns the value of a header field from the specified message part.
GetHeaders	Overloaded. Retrieves the headers for the specified message from the server.
GetMessage	Overloaded. Retrieve a message from the server and return the contents in a byte array.
GetType (inherited from Object)	Gets the Type of the current instance.
Idle	Overloaded. Enables mailbox status monitoring for the client session.
Initialize	Overloaded. Initialize an instance of the ImapClient class.
OpenMessage	Overloaded. Open the specified message for reading.
Read	Overloaded. Read data from the server and store it in a byte array.
Refresh	Updates the list of available mailboxes.
RenameMailbox	Change the name of a mailbox.
ReselectMailbox	Reselects the current mailbox.
Reset	Reset the internal state of the object, resetting all properties to their default values.
SearchMailbox	Overloaded. Search the current mailbox for messages that match the specified criteria and character set.
SelectMailbox	Selects the specified mailbox for read-write access.

StoreMessage	Overloaded. Retrieve a message from the current mailbox and store it in a file on the local system.
SubscribeMailbox	Overloaded. Subscribes the user to the specified mailbox.
ToString (inherited from Object)	Returns a String that represents the current Object.
UndeleteMessage	Removes the deletion flag for the specified message.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
UnselectMailbox	Overloaded. Unselects the current mailbox.
UnsubscribeMailbox	Overloaded. Unsubscribes the user from the specified mailbox.
Write	Overloaded. Write one or more bytes of data to the server.

Public Instance Events

OnCancel	Occurs when a blocking client operation is canceled.
OnConnect	Occurs when a connection is established with the remote host.
OnDisconnect	Occurs when the remote host disconnects from the local system.
OnError	Occurs when an client operation fails.
OnProgress	Occurs as a data stream is being read or written to the client.
OnRead	Occurs when data is available to be read from the client.
OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
OnUpdate	Occurs when the server sends a mailbox update notification to the client.
OnWrite	Occurs when data can be written to the client.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the ImapClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

ImapClient Constructor

Initializes a new instance of the ImapClient class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public ImapClient();
```

See Also

ImapClient Class | SocketTools Namespace

ImapClient Fields

The fields of the **ImapClient** class are listed below. For a complete list of **ImapClient** class members, see the ImapClient Members topic.

Public Static (Shared) Fields

imapPortDefault	A constant value which specifies the default port number.
imapPortSecure	A constant value which specifies the default port number for a secure connection.
imapTimeout	A constant value which specifies the default timeout period.

Public Instance Fields

Mailbox	Gets the names of the available mailboxes for the current user.
MessagePart	Gets the contents of the specified message part.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Mailbox Field

Gets the names of the available mailboxes for the current user.

[Visual Basic]

```
Public ReadOnly Mailbox As MailboxArray
```

[C#]

```
public readonly MailboxArray Mailbox;
```

Remarks

The **Mailbox** array is used to enumerate the available mailboxes on the IMAP server. This is a zero-based array, which means that the index value for the first mailbox is zero. The total number of mailboxes that are available on the server is returned by the **Mailboxes** property.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.MessagePart Field

Gets the contents of the specified message part.

[Visual Basic]
```
Public ReadOnly MessagePart As MessagePartArray
```

[C#]
```
public readonly MessagePartArray MessagePart;
```

Remarks

The **MessagePart** array returns the contents of the specified message part. All messages have at least one part, which consists of one or more header fields, followed by the body of the message. The default part, part 1, refers to the main message header and body. If the message contains multiple parts (as with a message that contains one or more attached files), the **MessagePart** array can be set to refer to that specific part of the message.

Messages with file attachments typically consist of a message part which describes the contents of the attachment, followed by the attachment itself. For a message with one attached file, there would be a total of three parts. Part 1 would refer to the main message part, which contains the headers such as From, To, Subject, Date and so on. For multipart messages, part 1 typically does not have a message body, since any text is usually created as a separate part (for those messages that do not contain multiple parts, the part 1 body contains the text message). Part 2 would contain the text describing the attachment, and part 3 would contain the attachment itself. If the attached file is binary, then the transfer encoding type would usually be base64.

It is important to note that an IMAP server considers the first part of a multipart message to be part 1, so the **MessagePart** array is one-based. This is different than the **SocketTools.MailMessage** class, which considers the first part of a mulitpart message to be zero

See Also

ImapClient Class | SocketTools Namespace

ImapClient Properties

The properties of the **ImapClient** class are listed below. For a complete list of **ImapClient** class members, see the ImapClient Members topic.

Public Instance Properties

Authentication	Gets and sets the method used to authenticate the user.
AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
BearerToken	Gets and sets the bearer token used with OAuth 2.0 authentication.
Blocking	Gets and sets a value which indicates if the client is in blocking mode.
CertificateExpires	Get a value that specifies the date that the security certificate expires.
CertificateIssued	Get a value that specifies the date that the security certificate was issued.
CertificateIssuer	Get a value that provides information about the organization that issued the certificate.
CertificateName	Gets and sets a value that specifies the name of the client certificate.
CertificatePassword	Gets and sets the password associated with the client certificate.
CertificateStatus	Gets a value which indicates the status of the security certificate returned by the remote host.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateSubject	Gets a value that provides information about the organization that the server certificate was issued to.
CertificateUser	Gets and sets the user that owns the client certificate.
CipherStrength	Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.
Delimiter	Gets the hierarchical path delimiter used for the current mailbox.
Handle	Gets a value that specifies the client handle allocated for the current session.
HashStrength	Gets a value which specifies the length of the message digest that was selected for a secure connection.

HeaderField	Gets and sets the current header field name.
HeaderValue	Gets the value of the current header field.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsConnected	Gets a value which indicates if a connection to the remote host has been established.
IsIdle	Gets a value which indicates if the client is idle and the current mailbox is being monitored for status changes.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsReadable	Gets a value which indicates if there is data available to be read from the socket connection to the server.
IsWritable	Gets a value which indicates if data can be written to the client without blocking.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets the local Internet address that the client is bound to.
Localize	Gets a value that specifies if the date and time are localized.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
Mailboxes	Gets the number of mailboxes available on the server.
MailboxFlags	Gets one or more flags which identify characteristics of the current mailbox.
MailboxMask	Gets and sets the current mailbox wildcard mask.
MailboxName	Gets and sets the name of the current mailbox.
MailboxPath	Gets and sets the current mailbox reference path.
MailboxSize	Gets the size of the current mailbox.
MailboxUID	Gets the unique identifier for the current mailbox.
Message	Gets and sets the current message number.

MessageCount	Gets the number of messages available in the current mailbox.
MessageFlags	Gets and sets one or more flags for the current message.
MessageParts	Gets the number of message parts in the current message.
MessageSize	Gets the size of the current message in bytes.
MessageUID	Gets the UID for the current message.
NewMessages	Gets the number of new messages available in the current mailbox.
Options	Gets and sets a value which specifies one or more client options.
Password	Gets and sets the password used to authenticate the client.
ReadOnly	Gets a value which specifies if the current mailbox can be modified.
RecentMessages	Gets the number of messages which have recently arrived in the mailbox.
RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.
ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.
Secure	Gets and sets a value which specifies if a secure connection is established.
SecureCipher	Gets a value that specifies the encryption algorithm used for a secure connection.
SecureHash	Gets a value that specifies the message digest algorithm used for a secure connection.
SecureKeyExchange	Gets a value that specifies the key exchange algorithm used for a secure connection.
SecureProtocol	Gets and sets a value which specifies the protocol used for a secure connection.
Status	Gets a value which specifies the current status of the client.
Subscribed	Gets a value that specifies if the user has subscribed to the currently selected mailbox.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.

ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
TimeZone	Gets and sets the current timezone offset in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
UserName	Gets and sets the username used to authenticate the client.
Version	Gets a value which returns the current version of the ImapClient class library.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Authentication Property

Gets and sets the method used to authenticate the user.

[Visual Basic]

```
Public Property Authentication As ImapAuthentication
```

[C#]

```
public ImapClient.ImapAuthentication Authentication {get; set;}
```

Property Value

A ImapAuthentication enumeration value which specifies the authentication method.

Remarks

The **authXOAuth2** and **authBearer** authentication methods are similar, but they are not interchangeable. Both use an OAuth 2.0 bearer token to authenticate the client session, but they differ in how the token is presented to the server. It is currently preferable to use the XOAUTH2 method because it is more widely available and some service providers do not yet support the OAUTHBEARER method.

See Also

ImapClient Class | SocketTools Namespace | BearerToken Poperty | Password Property | UserName Property

ImapClient.AutoResolve Property

Gets and sets a value that determines if host names and addresses are automatically resolved.

[Visual Basic]

```
Public Property AutoResolve As Boolean
```

[C#]

```
public bool AutoResolve {get; set;}
```

Property Value

Returns **true** if host names are automatically resolved to Internet addresses. The default value is **false**.

Remarks

Setting the **AutoResolve** property determines if the class automatically resolves host names and addresses specified by the **HostName** and **HostAddress** properties. If set to **true**, setting the **HostName** property will cause the class to automatically determine the corresponding IP address and update the **HostAddress** property accordingly. Likewise, setting the **HostAddress** property will cause the class to determine the host name and update the **HostName** property. Setting this property to **false** prevents the class from resolving host names until a connection attempt is made.

It is important to note that setting the **HostName** or **HostAddress** property may cause the current thread to block, sometimes for several seconds, until the name or address is resolved. To prevent this behavior, set this property value to **false**.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.BearerToken Property

Gets and sets the bearer token used with OAuth 2.0 authentication.

[Visual Basic]
```
Public Property BearerToken As String
```

[C#]
```
public string BearerToken {get; set;}
```

Property Value

Returns a string which contains the bearer token. Assigning a value to this property sets the curent authentication type to use OAuth 2.0 authentication and updates the bearer token.

Remarks

Assigning a value to the **BearerToken** property will automatically change the current authentication method to use OAuth 2.0 if necessary.

You should only use an OAuth 2.0 authentication method if you understand the process of how to request the access token. Obtaining a bearer token requires registering your application with the mail service provider (e.g.: Microsoft or Google), getting a unique client ID associated with your application and then requesting the bearer token using the appropriate scope for the service. Obtaining the initial token will typically involve interactive confirmation on the part of the user, requiring they grant permission to your application to access their mail account.

Your application should not store the bearer token for later use. They usually have a relatively short lifespan, typically about an hour, and are designed to be used with the current client session. You should specify offline access as part of the OAuth 2.0 scope, and store the refresh token provided by the service. The refresh token has a much onger validity period and can be used to obtain a new access token when needed.

If the current authentication method does not use OAuth 2.0, this property will return an empty string and you should use the **Password** property to obtain the current user password.

See Also

ImapClient Class | SocketTools Namespace | Authentication Property | Password Property | UserName Property

ImapClient.Blocking Property

Gets and sets a value which indicates if the client is in blocking mode.

[Visual Basic]

```
Public Property Blocking As Boolean
```

[C#]

```
public bool Blocking {get; set;}
```

Property Value

Returns **true** if the client is in blocking mode; otherwise it returns **false**. The default value is **true**.

Remarks

Setting the **Blocking** property determines if client operations complete synchronously or asynchronously. If set to **true**, then each client operation (such as sending or receiving data) will return when the operation has completed or timed-out. If set to **false**, client operations will return immediately. If the operation would result in the client blocking (such as attempting to read data when no data has been sent by the remote host), an error is generated.

It is important to note that certain events, such as **OnDisconnect, OnRead** and **OnWrite** are only fired if the client is in non-blocking mode.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.CertificateExpires Property

Get a value that specifies the date that the security certificate expires.

[Visual Basic]

```
Public ReadOnly Property CertificateExpires As String
```

[C#]

```
public string CertificateExpires {get;}
```

Property Value

A string which specifies a date using the local date and time format.

Remarks

The **CertificateExpires** property returns a string that specifies the date and time that the security certificate expires. This property will return an empty string if a secure connection has not been established with the remote host.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.CertificateIssued Property

Get a value that specifies the date that the security certificate was issued.

[Visual Basic]

```
Public ReadOnly Property CertificateIssued As String
```

[C#]

```
public string CertificateIssued {get;}
```

Property Value

A string which specifies a date using the local date and time format.

Remarks

The **CertificateIssued** property returns a string that specifies the date and time that the security certificate was issued. This property will return an empty string if a secure connection has not been established with the remote host.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.CertificateIssuer Property

Get a value that provides information about the organization that issued the certificate.

[Visual Basic]

```
Public ReadOnly Property CertificateIssuer As String
```

[C#]

```
public string CertificateIssuer {get;}
```

Property Value

A string that contains a comma separated list of name value pairs.

Remarks

The **CertificateIssuer** property returns a string that contains information about the organization that issued the server certificate. The string value is a comma separated list of tagged name and value pairs. In the nomenclature of the X.500 standard, each of these pairs are called a relative distinguished name (RDN), and when concatenated together, forms the issuer's distinguished name (DN). For example:

```
C=US, O="RSA Data Security, Inc.", OU=Secure Server Certification Authority
```

To obtain a specific value, such as the name of the issuer or the issuer's country, the application must parse the string returned by this property. Some of the common tokens used in the distinguished name are:

Token	Description
C	The ISO standard two character country code.
S	The name of the state or province.
L	The name of the city or locality.
O	The name of the company or organization.
OU	The name of the department or organizational unit
CN	The common name; with X.509 certificates, this is the domain name of the site the certificate was issued for.

This property will return an empty string if a secure connection has not been established with the remote host.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.CertificateName Property

Gets and sets a value that specifies the name of the client certificate.

[Visual Basic]

```
Public Property CertificateName As String
```

[C#]

```
public string CertificateName {get; set;}
```

Property Value

A string which specifies the certificate name.

Remarks

The **CertificateName** property is used to specify the name of a client certificate to use when establishing a secure connection. It is only required that you set this property value if the server requires a client certificate for authentication. If this property is not set, a client certificate will not be provided to the server. If a certificate name is specified, the certificate must have a private key associated with it, otherwise the connection attempt will fail because the control will be unable to create a security context for the session.

When the certificate store is searched for a matching certificate, it will first search for any certificate with a friendly name that matches the property value. If no valid certificate is found, it will then search for a certificate with a matching common name.

Certificates may be installed and viewed on the local system using the Certificate Manager that is included with the Windows operating system. For more information, refer to the documentation for the Microsoft Management Console.

See Also

ImapClient Class | SocketTools Namespace | CertificateStore Property | Secure Property

ImapClient.CertificateStatus Property

Gets a value which indicates the status of the security certificate returned by the remote host.

[Visual Basic]

```
Public ReadOnly Property CertificateStatus As SecurityCertificate
```

[C#]

```
public ImapClient.SecurityCertificate CertificateStatus {get;}
```

Property Value

A SecurityCertificate enumeration value which specifies the status of the certificate.

Remarks

The **CertificateStatus** property is used to determine the status of the security certificate returned by the remote host when a secure connection has been established. This property value should be checked after the connection to the server has completed, but prior to beginning a transaction.

Note that if the certificate cannot be validated, the secure connection will not be automatically terminated. It is the responsibility of your application to determine the best course of action to take if the certificate is invalid. Even if the security certificate cannot be validated, the data exchanged with the remote host will still be encrypted.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.CertificateStore Property

Gets and sets a value that specifies the name of the local certificate store.

[Visual Basic]
```
Public Property CertificateStore As String
```

[C#]
```
public string CertificateStore {get; set;}
```

Property Value

A string which specifies the certificate store name. The default value is the current user's personal certificate store.

Remarks

The **CertificateStore** property is used to specify the name of the certificate store which contains the security certificate to use when establishing a secure connection. The certificate may either be stored in the registry or in a file. If the certificate is stored in the registry, then this property should be set to one of the following predefined values:

Store Name	Description
CA	Certification authority certificates. These are certificates that are issued by entities which are entrusted to issue certificates to other individuals or organizations. Companies such as VeriSign and Thawte act as certification authorities.
MY	Personal certificates and their associated private keys for the current user. This store typically holds the client certificates used to establish a user's credentials. If a certificate store is not specified, this is the default value that is used.
ROOT	Certificates that have been self-signed by a certificate authority. Root certificates for a number of different certification authorities such as VeriSign and Thawte are installed as part of the operating system and periodically updated by Microsoft.

In most cases the client certificate will be installed in the user's personal certificate store, and therefore it is not necessary to set this property value because that is the default location that will be used to search for the certificate. This property is only used if the **CertificateName** property is also set to a valid certificate name.

If you are using a local certificate store, with the certificate and private key stored in the registry, you can explicitly specify whether the certificate store for the current user or the local machine (all users) should be used. This is done by prefixing the certificate store name with "HKCU" for the current user, or "HKLM" for the local machine. For example, a certificate store name of "HKLM:MY" would specify the personal certificate store for the local machine, rather than the current user. If neither prefix is specified, it will default to the certificate store for the current user.

This property may also be used to specify a file that contains the client certificate. In this case, the property should specify the full path to the file and must contain both the certificate and private key in PKCS12

format. If the file is protected by a password, the **CertificatePassword** property must also be set to specify the password.

See Also

ImapClient Class | SocketTools Namespace | CertificatePassword Property | Secure Property

ImapClient.CertificateSubject Property

Gets a value that provides information about the organization that the server certificate was issued to.

[Visual Basic]

```
Public ReadOnly Property CertificateSubject As String
```

[C#]

```
public string CertificateSubject {get;}
```

Property Value

A string that contains a comma separated list of name value pairs.

Remarks

The **CertificateSubject** property returns a string that contains information about the organization that the server certificate was issued to. The string value is a comma separated list of tagged name and value pairs. In the nomenclature of the X.500 standard, each of these pairs are called a relative distinguished name (RDN), and when concatenated together, forms the issuer's distinguished name (DN). For example:

```
C=US, O="RSA Data Security, Inc.", OU=Secure Server Certification Authority
```

To obtain a specific value, such as the name of the issuer or the issuer's country, the application must parse the string returned by this property. Some of the common tokens used in the distinguished name are:

Token	Description
C	The ISO standard two character country code.
S	The name of the state or province.
L	The name of the city or locality.
O	The name of the company or organization.
OU	The name of the department or organizational unit
CN	The common name; with X.509 certificates, this is the domain name of the site the certificate was issued for.

This property will return an empty string if a secure connection has not been established with the remote host.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.CipherStrength Property

Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.

[Visual Basic]
```
Public ReadOnly Property CipherStrength As Integer
```

[C#]
```
public int CipherStrength {get;}
```

Property Value

An integer value which specifies the encryption key length if a secure connection has been established; otherwise a value of 0 is returned.

Remarks

The **CipherStrength** property returns the number of bits in the key used to encrypt the secure data stream. Common values returned by this property are 128 and 256. A key length of 40 or 56 bits is considered insecure and subject to brute force attacks. 128-bit and 256-bit keys are considered secure. If this property returns a value of 0, this means that a secure connection has not been established with the remote host.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Delimiter Property

Gets the hierarchical path delimiter used for the current mailbox.

[Visual Basic]

```
Public ReadOnly Property Delimiter As String
```

[C#]

```
public string Delimiter {get;}
```

Property Value

A string which specifies the path delimiter used for the current mailbox.

Remarks

The **Delimiter** property returns a string which specifies the path delimiter used for the current mailbox. If the IMAP server supports multiple levels of mailboxes, then a special character or sequence of characters are used as delimiters between different levels of the mailbox hierarchy. On most systems, including most UNIX and Windows platforms, the delimiter is the forward slash "/" character.

It is possible that an IMAP server may only support a flat namespace, in which case this property will return an empty string.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Handle Property

Gets a value that specifies the client handle allocated for the current session.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer which represents a client handle. If there is no active connection, a value of -1 is returned.

Remarks

The **Handle** property specifies the numeric descriptor of the current client session.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.HashStrength Property

Gets a value which specifies the length of the message digest that was selected for a secure connection.

[Visual Basic]

```
Public ReadOnly Property HashStrength As Integer
```

[C#]

```
public int HashStrength {get;}
```

Property Value

An integer value which specifies the length of the message digest if a secure connection has been established; otherwise a value of 0 is returned.

Remarks

The **HashStrength** property returns the number of bits used in the message digest (hash) that was selected. Common values returned by this property are 128 and 160. If this property returns a value of 0, this means that a secure connection has not been established with the remote host.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.HeaderField Property

Gets and sets the current header field name.

[Visual Basic]

```
Public Property HeaderField As String
```

[C#]

```
public string HeaderField {get; set;}
```

Property Value

A string which specifies the current header field name.

Remarks

The **HeaderField** property returns the name of the current header field. Setting this property causes the control to determine if that header field exists, and if it does, to update the **HeaderValue** property with its value. This property can be used to obtain the value of any header in the current message.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.HeaderValue Property

Gets the value of the current header field.

[Visual Basic]

```
Public ReadOnly Property HeaderValue As String
```

[C#]

```
public string HeaderValue {get;}
```

Property Value

A string which specifies the value of the current header field.

Remarks

The **HeaderValue** property returns the value of the header field specified by the **HeaderField** property. This property can be used to obtain the value of any header in the current message.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.HostAddress Property

Gets and sets a value which specifies the Internet address used to establish a connection.

[Visual Basic]
```
Public Property HostAddress As String
```

[C#]
```
public string HostAddress {get; set;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **HostAddress** property can be used to set the Internet address for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the address is assigned to a valid host name, the **HostName** property will be updated with that value.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.HostName Property

Gets and sets a value which specifies the host name used to establish a connection.

[Visual Basic]

```
Public Property HostName As String
```

[C#]

```
public string HostName {get; set;}
```

Property Value

A string which specifies a host name.

Remarks

The **HostName** property can be used to set the host name for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the name can be resolved to a valid Internet address, the **HostAddress** property will be updated with that value.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.IsBlocked Property

Gets a value which indicates if the current thread is performing a blocking client operation.

[Visual Basic]
```
Public ReadOnly Property IsBlocked As Boolean
```

[C#]
```
public bool IsBlocked {get;}
```

Property Value

Returns **true** if the current thread is blocking, otherwise returns **false**.

Remarks

The **IsBlocked** property returns **true** if the current thread is blocked performing an operation. Because the Windows Sockets API only permits one blocking operation per thread of execution, this property should be checked before starting any blocking operation in response to an event.

If the **IsBlocked** property returns **false**, this means there are no blocking operations on the current thread at that time. However, this does not guarantee that the next client operation will not fail. An application should always check the return value from a client operation and check the value of the **LastError** property if an error occurs.

Note that this property will return **true** if there is any blocking operation being performed by the current thread, regardless of whether this specific instance of the class is responsible for the blocking operation or not.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.IsConnected Property

Gets a value which indicates if a connection to the remote host has been established.

[Visual Basic]

```
Public ReadOnly Property IsConnected As Boolean
```

[C#]

```
public bool IsConnected {get;}
```

Property Value

Returns **true** if the connection has been established; otherwise returns **false**.

Remarks

The **IsConnected** property can only be used to indicate if there is still a logical connection to the remote host. It cannot be used to detect abnormal conditions such as the remote host aborting the connection, the physical network connection being lost or other critical errors.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.IsIdle Property

Gets a value which indicates if the client is idle and the current mailbox is being monitored for status changes.

[Visual Basic]

```
Public ReadOnly Property IsIdle As Boolean
```

[C#]

```
public bool IsIdle {get;}
```

Property Value

Returns **true** if the client is idle and the current mailbox is being monitored; otherwise returns **false**.

Remarks

The **IsIdle** property can be used to determine if the **Idle** method has been called to place the client session in an idle state, monitoring the connection for any status messages sent by the server. Typically this is done to allow the application to be notified asynchronously whenever a new message is stored in the mailbox, or when a message has been expunged.

The worker thread that monitors the client connection in the background can terminate if an IMAP command is sent to the server, if the **Cancel** method is called or if the client disconnects from the server. This property enables the application to determine if this background thread is still active or not.

See Also

ImapClient Class | SocketTools Namespace | Idle Method | OnUpdate Event

ImapClient.IsInitialized Property

Gets a value which indicates if the current instance of the class has been initialized successfully.

[Visual Basic]

```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]

```
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the class instance has been initialized; otherwise returns **false**.

Remarks

The **IsInitialized** property is used to determine if the current instance of the class has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the class may not be able to initialize itself.

The most common reasons that a class instance may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required networking libraries or not being able to access the system registry.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.IsReadable Property

Gets a value which indicates if there is data available to be read from the socket connection to the server.

[Visual Basic]

```
Public ReadOnly Property IsReadable As Boolean
```

[C#]

```
public bool IsReadable {get;}
```

Property Value

Returns **true** if there is data available to be read; otherwise returns **false**.

Remarks

The **IsReadable** property returns **true** if data can be read from the client without blocking. For non-blocking sessions, this property can be checked before the application attempts to read the client. Note that even if this property does return **true** indicating that there is data available to be read, applications should always check the return value from the **Read** method.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.IsWritable Property

Gets a value which indicates if data can be written to the client without blocking.

[Visual Basic]

```
Public ReadOnly Property IsWritable As Boolean
```

[C#]

```
public bool IsWritable {get;}
```

Property Value

Returns **true** if data can be written to the client; otherwise returns **false**.

Remarks

The **IsWritable** property returns **true** if data can be written to the client without blocking. For non-blocking sessions, this property can be checked before the application attempts to write data to the client. Note that even if this property does return **true** indicating that data can be written to the client, applications should always check the return value from the **Write** method.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public ImapClient.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.LocalAddress Property

Gets the local Internet address that the client is bound to.

[Visual Basic]

```
Public ReadOnly Property LocalAddress As String
```

[C#]

```
public string LocalAddress {get;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **LocalAddress** property returns the local Internet address that the client is bound to when a connection is established with a remote host. This property may return either an IPv4 or IPv6 formatted address, depending on the type of connection that was established.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Localize Property

Gets a value that specifies if the date and time are localized.

[Visual Basic]

```
Public Property Localize As Boolean
```

[C#]

```
public bool Localize {get; set;}
```

Property Value

A boolean value which specifies if the date and time is localized.

Remarks

Setting the **Localize** property controls how date and time values are localized. If the property is set to **true**, then the date and time will be adjusted to the current timezone. If the property is set to **false**, the date and time are returned as UTC (Coordinated Universal Time) values.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.LocalName Property

Gets a value which specifies the host name for the local system.

[Visual Basic]

```
Public ReadOnly Property LocalName As String
```

[C#]

```
public string LocalName {get;}
```

Property Value

A string which specifies the local host name.

Remarks

The **LocalName** property returns the fully-qualified host name assigned to the local system. If the system has not been configured with an Internet domain name, then this property will return the NetBIOS name assigned to the local system.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.LocalPort Property

Gets the local port number the client is bound to.

[Visual Basic]

```
Public ReadOnly Property LocalPort As Integer
```

[C#]

```
public int LocalPort {get;}
```

Property Value

An integer value which specifies a port number. The default value is 0.

Remarks

The **LocalPort** property is used to identify the local port number that the client is bound to to when a connection is established with a remote host.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Mailboxes Property

Gets the number of mailboxes available on the server.

[Visual Basic]

```
Public ReadOnly Property Mailboxes As Integer
```

[C#]

```
public int Mailboxes {get;}
```

Property Value

An integer value which specifies the number of available mailboxes.

Remarks

The **Mailboxes** property returns the total number of mailboxes available to the current account on the server. This property can be used in conjunction with the **Mailbox** array to enumerate the names of all of the mailboxes which can be selected by the client.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.MailboxFlags Property

Gets one or more flags which identify characteristics of the current mailbox.

[Visual Basic]

```
Public ReadOnly Property MailboxFlags As ImapFlags
```

[C#]

```
public ImapClient.ImapFlags MailboxFlags {get;}
```

Property Value

An ImapFlags enumeration value which specifies one or more mailbox flags.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.MailboxMask Property

Gets and sets the current mailbox wildcard mask.

[Visual Basic]

```
Public Property MailboxMask As String
```

[C#]

```
public string MailboxMask {get; set;}
```

Property Value

A string which specifies the current mailbox wildcard mask.

Remarks

The **MailboxMask** property returns the current mailbox wildcard mask. If no wildcard mask has been specified by the client, this property will return an empty string.

Setting the **MailboxMask** property will determine which mailboxes are returned by the **Mailbox** array. Wildcards may include the asterisk (which matches any mailbox as well as any child mailboxes) and the percent sign (which matches any mailbox, but does not match any child mailboxes). This property may be used in conjunction with the **MailboxPath** property to further qualify which mailboxes are returned.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.MailboxName Property

Gets and sets the name of the current mailbox.

[Visual Basic]

```
Public Property MailboxName As String
```

[C#]

```
public string MailboxName {get; set;}
```

Property Value

A string that specifies the name of the current mailbox.

Remarks

The **MailboxName** property returns the name of the currently selected mailbox. If no mailbox has been selected by the client, this property will return an empty string.

Setting the **MailboxName** property will select a new mailbox in read-write mode. If the client has a different mailbox currently selected, that mailbox will be closed and any messages marked for deletion will be expunged. To prevent deleted messages from being removed from the previous mailbox, call the **UnselectMailbox** method prior to selecting the new mailbox. Setting the **MailboxName** property to an empty string will cause the current mailbox to be unselected, and a new mailbox will not be selected. Before the application can access any messages, it must select a new mailbox.

Selecting a new mailbox will automatically update those properties which provide information about the current mailbox, such as the **MailboxFlags** and **MailboxUID** properties. If an application wishes to update the information for the current mailbox, simply set the **MailboxName** property again with the same mailbox name. Note that this will not cause any messages marked for deletion to be expunged.

The special case-insensitive mailbox name INBOX is used for new messages. Other mailbox names may or may not be case-sensitive depending on the IMAP server's operating system and implementation.

If the mailbox name contains international characters then it is automatically encoded using a modified version of UTF-7 encoding. For example, if a mailbox is named "Håndskrift", the mailbox name created on the server will be the string "H&AOU-ndskrift". The control will automatically decode UTF-7 encoded mailbox names, making the conversion transparent to the application.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.MailboxPath Property

Gets and sets the current mailbox reference path.

[Visual Basic]
```
Public Property MailboxPath As String
```

[C#]
```
public string MailboxPath {get; set;}
```

Property Value

A string which specifies the current mailbox reference path.

Remarks

The **MailboxPath** property returns the current mailbox reference path. If no path has been specified by the client, this property will return an empty string.

Setting the **MailboxPath** property will determine which mailboxes are returned by the **Mailbox** array. Typically this is used to specify a subdirectory where mail folders are stored for the current user. Note that some mail servers may not permit absolute reference paths, and in most cases the path should include a trailing slash. This property may be used in conjunction with the **MailboxMask** property to further qualify which mailboxes are returned.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.MailboxSize Property

Gets the size of the current mailbox.

[Visual Basic]

```
Public ReadOnly Property MailboxSize As Integer
```

[C#]

```
public int MailboxSize {get;}
```

Property Value

An integer value which specifies the size of the mailbox in bytes.

Remarks

The **MailboxSize** property returns the combined size of all messages in the current mailbox. Referencing this property will cause the current thread to block and may require a significant amount of time to calculate the mailbox size if there are a large number of messages in the mailbox. Because it can potentially result in long delays, it is not recommended that an application calculate the mailbox size unless it is absolutely necessary.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.MailboxUID Property

Gets the unique identifier for the current mailbox.

[Visual Basic]

```
Public ReadOnly Property MailboxUID As Integer
```

[C#]

```
public int MailboxUID {get;}
```

Property Value

An integer value which specifies the mailbox UID.

Remarks

The **MailboxUID** property returns an integer value which uniquely identifies the mailbox and corresponds to the UIDVALIDITY value returned by the IMAP server. The actual value is determined by the server and should be considered opaque. The protocol specification requires that a mailbox's UID must not change unless the mailbox contents are modified or existing messages in the mailbox have been assigned new UIDs.

An application can use the **MailboxUID** property value in combination with the **MessageUID** property in order to uniquely identify a message on the server. However, the application must take into consideration that the IMAP server can reassign new message UIDs when the mailbox is modified. If the mailbox and message UIDs are being stored on the local system to track what messages have been retrieved from the server, the application must check the UID of the mailbox whenever it is selected. If the mailbox UID has changed, this means that the UIDs for the messages in that mailbox may have changed. The client should resynchronize with the server, and update it's local copy of that mailbox.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Message Property

Gets and sets the current message number.

[Visual Basic]

```
Public Property Message As Integer
```

[C#]

```
public int Message {get; set;}
```

Property Value

An integer value which specifies the current message number.

Remarks

The **Message** property sets or returns the message number for the currently selected mailbox. Message numbers range from 1 through the number of messages available on the server, as returned by the **MessageCount** property. Setting the **Message** property to an invalid message number will generate an exception.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.MessageCount Property

Gets the number of messages available in the current mailbox.

[Visual Basic]

```
Public ReadOnly Property MessageCount As Integer
```

[C#]

```
public int MessageCount {get;}
```

Property Value

An integer value which specifies the number of messages.

Remarks

The **MessageCount** property returns the number of messages available to be retrieved from the currently selected mailbox.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.MessageFlags Property

Gets and sets one or more flags for the current message.

[Visual Basic]
```
Public Property MessageFlags As ImapFlags
```

[C#]
```
public ImapClient.ImapFlags MessageFlags {get; set;}
```

Property Value

An ImapFlags enumeration value which specifies one or more message flags.

Remarks

The **MessageFlags** property returns information about the currently selected message specified by the **Message** property. Setting the **MessageFlags** property changes the flags for the currently selected message. Multiple bit flags can be combined using the bitwise Or operator. An application can test if a flag is set by using the bitwise And operator.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.MessageParts Property

Gets the number of message parts in the current message.

[Visual Basic]
```
Public ReadOnly Property MessageParts As Integer
```

[C#]
```
public int MessageParts {get;}
```

Property Value

An integer value which specifies the number of message parts.

Remarks

The **MessageParts** property returns the number of message parts in the current message. All messages have at least one part, referenced as part 1. Multipart messages will consist of additional parts which may be accessed by reading the **MessagePart** array or calling the **GetMessage** method.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.MessageSize Property

Gets the size of the current message in bytes.

[Visual Basic]

```
Public ReadOnly Property MessageSize As Integer
```

[C#]

```
public int MessageSize {get;}
```

Property Value

An integer value which specifies the size of the message.

Remarks

The **MessageSize** property returns the size of the current message in bytes. The size includes the header and body portion of the message.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.MessageUID Property

Gets the UID for the current message.

[Visual Basic]

```
Public ReadOnly Property MessageUID As Integer
```

[C#]

```
public int MessageUID {get;}
```

Property Value

An integer value which specifies the current message UID.

Remarks

The **MessageUID** property returns an integer value which specifies a unique identifier for this message. The actual value is determined by the server and should be considered opaque. If the client application stores the message UID on the local system, it should also store the UID for the mailbox that contains the message. If the mailbox UID changes, the message UID may no longer be valid.

An application can use the **MessageUID** property value in combination with the **MailboxUID** property in order to uniquely identify a message on the server. However, the application must take into consideration that the IMAP server can reassign new message UIDs when the mailbox is modified. If the mailbox and message UIDs are being stored on the local system to track what messages have been retrieved from the server, the application must check the UID of the mailbox whenever it is selected. If the mailbox UID has changed, this means that the UIDs for the messages in that mailbox may have changed. The client should resynchronize with the server, and update it's local copy of that mailbox.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.NewMessages Property

Gets the number of new messages available in the current mailbox.

[Visual Basic]

```
Public ReadOnly Property NewMessages As Integer
```

[C#]

```
public int NewMessages {get;}
```

Property Value

An integer value which specifies the number of new, unread messages in the current mailbox.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Options Property

Gets and sets a value which specifies one or more client options.

[Visual Basic]

```
Public Property Options As ImapOptions
```

[C#]

```
public ImapClient.ImapOptions Options {get; set;}
```

Property Value

Returns one or more ImapOptions enumeration flags which specify the options for the client. The default value for this property is **imapOptionNone**.

Remarks

The **Options** property specifies one or more default options options which are used when establishing a connection using the **Connect** method.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Password Property

Gets and sets the password used to authenticate the client.

[Visual Basic]

```
Public Property Password As String
```

[C#]

```
public string Password {get; set;}
```

Property Value

A string which specifies the password.

Remarks

The **Password** property specifies the password used to authenticate the client session. This property is used as the default value for the **Connect** method if no password is specified as an argument.

Refer to the **Authentication** property for more information on the available authentication methods. If you are using the OAuth 2.0 authentication method, this property should not be set to the user's password. Instead, you should set the **BearerToken** property to the OAuth 2.0 bearer token issued by the mail service provider. Note that these access tokens can be much larger than your typical password and are only valid for a limited period of time.

You can use the **Password** property to specify an OAuth 2.0 bearer token. However, it is recommended that you use the **BearerToken** property instead of assigning it to this property. It will ensure compatibility with future versions of the class and make it clear in your code you are using an OAuth 2.0 bearer token and not a password. If the **AuthType** property specifies one of the OAuth 2.0 authentication methods, this property will return the bearer token.

See Also

ImapClient Class | SocketTools Namespace | Authentication Property | BearerToken Property | UserName Property | Connect Method

ImapClient.ReadOnly Property

Gets a value which specifies if the current mailbox can be modified.

[Visual Basic]

```
Public ReadOnly Property ReadOnly As Boolean
```

[C#]

```
public bool ReadOnly {get;}
```

Property Value

A boolean value which specifies if the current mailbox can be modified. A value of **true** specifies that the mailbox cannot be modified by the client. A value of **false** specifies that the mailbox can be modified.

Remarks

The **ExamineMailbox** method can be used to select a mailbox in read-only mode. The **SelectMailbox** method can be used to select a mailbox in read-write mode, which allows the contents of the mailbox to be modified.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.RecentMessages Property

Gets the number of messages which have recently arrived in the mailbox.

[Visual Basic]

```
Public ReadOnly Property RecentMessages As Integer
```

[C#]

```
public int RecentMessages {get;}
```

Property Value

An integer value which specifies the number of recent messages.

Remarks

The **RecentMessages** property returns the number of messages which have been recently added to the currently selected mailbox. This property is particularly useful when the INBOX mailbox is selected because it enables the application to check if any new messages have arrived.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.RemotePort Property

Gets and sets a value which specifies the remote port number.

[Visual Basic]
```
Public Property RemotePort As Integer
```

[C#]
```
public int RemotePort {get; set;}
```

Property Value

An integer value which specifies a port number.

Remarks

The **RemotePort** property is used to set the port number that will be used to establish a connection with a remote host. If the port number specifies a well-known port, the **RemoteService** property will be updated with that name.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.ResultCode Property

Gets a value which specifies the last result code returned by the server.

[Visual Basic]
```
Public ReadOnly Property ResultCode As ImapResult
```

[C#]
```
public ImapClient.ImapResult ResultCode {get;}
```

Property Value

An ImapResult enumeration value which specifies the last result code returned by the server.

Remarks

One of the following result codes may be returned after a command is executed on the server:

ResultCode	Description
resultUnknown	An unknown result code was returned by the server.
resultOk	The previous command completed successfully. The result string contains information about the results of the command.
resultNo	The previous command could not be completed. The result string contains information about why the command failed.
resultBad	The previous command could not be completed, the command may be invalid or not supported on the server. The result string contains information about why the command failed.
resultContinue	The command has executed and is waiting for additional data from the client.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.ResultString Property

Gets a string value which describes the result of the previous command.

[Visual Basic]
```
Public ReadOnly Property ResultString As String
```

[C#]
```
public string ResultString {get;}
```

Property Value

A string which describes the result of the previous command executed on the server.

Remarks

The **ResultString** property returns the result string from the last action taken by the client. This string is generated by the remote server, and typically is used to describe the result code. For example, if the previous result code indicates an error, the result string may describe the condition that caused the error.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Secure Property

Gets and sets a value which specifies if a secure connection is established.

[Visual Basic]

```
Public Property Secure As Boolean
```

[C#]

```
public bool Secure {get; set;}
```

Property Value

Returns **true** if a secure connection is established; otherwise returns **false**. The default value is **false**.

Remarks

The **Secure** property determines if a secure connection is established with the remote host. The default value for this property is **false**, which specifies that a standard connection to the server is used. To establish a secure connection, the application should set this property value to **true** prior to calling the **Connect** method. Once the connection has been established, the client may exchange data with the server as with standard connections.

It is strongly recommended that any application that sets this property **true** use error handling to trap an errors that may occur. If the control is unable to initialize the security libraries, or otherwise cannot create a secure session for the client, an exception may be generated when this property value is set.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.SecureCipher Property

Gets a value that specifies the encryption algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureCipher As SecureCipherAlgorithm
```

[C#]

```
public ImapClient.SecureCipherAlgorithm SecureCipher {get;}
```

Property Value

A SecureCipherAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureCipher** property returns a value which identifies the algorithm used to encrypt the data stream. If a secure connection has not been established, this property will return a value of **cipherNone**.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.SecureHash Property

Gets a value that specifies the message digest algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureHash As SecureHashAlgorithm
```

[C#]

```
public ImapClient.SecureHashAlgorithm SecureHash {get;}
```

Property Value

A SecureHashAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureHash** property returns a value which identifies the message digest (hash) algorithm that was selected when a secure connection was established. If a secure connection has not been established, this property will return a value of **hashNone**.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.SecureKeyExchange Property

Gets a value that specifies the key exchange algorithm used for a secure connection.

[Visual Basic]
```
Public ReadOnly Property SecureKeyExchange As SecureKeyAlgorithm
```

[C#]
```
public ImapClient.SecureKeyAlgorithm SecureKeyExchange {get;}
```

Property Value

A SecureKeyAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureKeyExchange** property returns a value which identifies the key exchange algorithm that was selected when a secure connection was established. If a secure connection has not been established, this property will return a value of **keyExchangeNone**.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.SecureProtocol Property

Gets and sets a value which specifies the protocol used for a secure connection.

[Visual Basic]

```
Public Property SecureProtocol As SecurityProtocols
```

[C#]

```
public ImapClient.SecurityProtocols SecureProtocol {get; set;}
```

Property Value

A SecurityProtocols enumeration value which identifies the protocol to be used when establishing a secure connection.

Remarks

The **SecureProtocol** property can be used to specify the security protocol to be used when establishing a secure connection with a server or accepting a secure connection from a client. By default, the class will attempt to use either SSL v3 or TLS v1 to establish the connection, with the appropriate protocol automatically selected based on the capabilities of the remote host. It is recommended that you only change this property value if you fully understand the implications of doing so. Assigning a value to this property will override the default protocol and force the class to attempt to use only the protocol specified.

Multiple security protocols may be specified by combining them using a bitwise **or** operator. After a connection has been established, this property will identify the protocol that was selected. Attempting to set this property after a connection has been established will result in an exception being thrown. This property should only be set after setting the **Secure** property to **true** and before calling the **Accept** or **Connect** methods.

In some cases, a server may only accept a secure connection if the TLS v1 protocol is specified. If the security protocol is not compatible with the server, then the connection will fail with an error indicating that the control is unable to establish a security context for the session. In this case, try assigning the property to **protocolTLS1** and attempt the connection again.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Status Property

Gets a value which specifies the current status of the client.

[Visual Basic]

```
Public ReadOnly Property Status As ImapStatus
```

[C#]

```
public ImapClient.ImapStatus Status {get;}
```

Property Value

A ImapStatus enumeration value which specifies the current client status.

Remarks

The **Status** property returns the current status of the client. This property can be used to check on blocking connections to determine if the client is interacting with the remote host before taking some action.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Subscribed Property

Gets a value that specifies if the user has subscribed to the currently selected mailbox.

[Visual Basic]

```
Public Property Subscribed As Boolean
```

[C#]

```
public bool Subscribed {get; set;}
```

Property Value

A boolean value that specifies if the user has subscribed to the current mailbox.

Remarks

The **Subscribed** property is used to determine if the current mailbox has been subscribed to by the user. If the property returns **false**, the server has indicated that the user has not subscribed to the mailbox. If the property returns **true**, the current mailbox is in the user's subscription list.

Setting the **Subscribed** property changes the subscription status of the current mailbox. Setting the property to **true** adds the mailbox to the user's list of subscribed mailboxes, while setting it to **false** removes the mailbox from the subscription list.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.ThreadModel Property

Gets and sets a value which specifies the threading model for the class instance.

[Visual Basic]

```
Public Property ThreadModel As ThreadingModel
```

[C#]

```
public ImapClient.ThreadingModel ThreadModel {get; set;}
```

Property Value

Returns one or more ThreadingModel enumeration value which specifies the threading model for the client. The default value for this property is **modelSingleThread**.

Remarks

The **ThreadModel** property specifies the threading model that is used by the class instance when a connection is established. The default value for this property is **modelSingleThread**, which specifies that only the thread that established the connection should be permitted to invoke methods. It is important to note that this threading model does not limit the application to a single thread of execution. When a session is established using the **Connect** method, that session is attached to the thread that created it. From that point on, until the session is terminated, only the owner may invoke methods in that instance of the class. The ownership of the class instance may be transferred from one thread to another using the **AttachThread** method.

Setting this property to **modelFreeThread** disables certain internal safety checks that are performed by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads. The application must ensure that no two threads will attempt to invoke a blocking method at the same time. In other words, if one thread invokes a method, the application must ensure that another thread will not attempt to invoke any other method at the same time using the same instance of the class.

Changing the value of this property will not affect an active client session. The threading model must be specified prior to invoking the **Connect** method.

See Also

ImapClient Class | SocketTools Namespace | AttachThread Method | ThreadingModel Enumeration | ThreadModel Attribute

ImapClient.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]

```
Public Property ThrowError As Boolean
```

[C#]

```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Timeout Property

Gets and sets a value which specifies a timeout period in seconds.

[Visual Basic]

```
Public Property Timeout As Integer
```

[C#]

```
public int Timeout {get; set;}
```

Property Value

An integer value which specifies a timeout period in seconds.

Remarks

Setting the **Timeout** property specifies the number of seconds until a blocking operation fails and returns an error.

The timeout period is only used when the client is in blocking mode. Although this property can be changed when the client is in non-blocking mode, the value will be ignored until the client is returned to blocking mode.

For most applications it is recommended the timeout period be set between 10 and 20 seconds.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.TimeZone Property

Gets and sets the current timezone offset in seconds.

[Visual Basic]

```
Public Property TimeZone As Integer
```

[C#]

```
public int TimeZone {get; set;}
```

Property Value

An integer value which specifies the current timezone offset in seconds.

Remarks

The **TimeZone** property returns the current offset from UTC in seconds. Setting the property changes the current timezone offset to the specified value. The value of this property is initially determined by the date and time settings on the local system.

This property value is used in conjunction with the **Localize** property to control how date and time localization is handled.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Trace Property

Gets and sets a value which indicates if network function logging is enabled.

[Visual Basic]

```
Public Property Trace As Boolean
```

[C#]

```
public bool Trace {get; set;}
```

Property Value

Returns **true** if network function tracing is enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Trace** property is used to enable (or disable) the tracing of network function calls. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the SocketTools networking classes and is enabled or disabled for an entire process. This means that once trace logging is enabled for a given component, all of the function calls made by the process using any of the SocketTools classes will be logged. For example, if you have an application using both the File Transfer Protocol and Post Office Protocol classes, and you set the **Trace** property to **true**, function calls made by both classes will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all classes used by the process.

If trace logging is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the logfile is fairly large. Since logfiles can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

When redistributing your application, make sure that you include the **SocketTools10.TraceLog.dll** module with your installation. If this library is not present, then no trace output will be generated and the value of the **Trace** property will be ignored. Only those function calls made by the SocketTools networking classes will be logged. Calls made directly to the Windows Sockets API, or calls made by other classes, will not be logged.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.TraceFile Property

Gets and sets a value which specifies the name of the logfile.

[Visual Basic]
```
Public Property TraceFile As String
```

[C#]
```
public string TraceFile {get; set;}
```

Property Value

A string which specifies the name of the file.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named **SocketTools.log** is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFile** property should be set to the same value for each instance. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
 MyApp INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0 MyApp WRN:
connect(46, 192.0.0.1:1234, 16) returned -1 [10035] MyApp ERR: accept(46,
NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the control is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Note that if the specified file cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.TraceFlags Property

Gets and sets a value which specifies the client function tracing flags.

[Visual Basic]
```
Public Property TraceFlags As TraceOptions
```

[C#]
```
public ImapClient.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration which specifies the amount of detail written to the trace logfile.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Because network function tracing is enabled per-process, the trace flags are shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFlags** property should be set to the same value for each instance. Changing the trace flags for any one instance of the class will affect the logging performed for all SocketTools classes used by the application.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being written and the error **errorOperationWouldBlock** occurs, a warning is generated because the application simply needs to attempt to write the data at a later time.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.UserName Property

Gets and sets the username used to authenticate the client.

[Visual Basic]

```
Public Property UserName As String
```

[C#]

```
public string UserName {get; set;}
```

Property Value

A string which specifies the username.

Remarks

The **UserName** property specifies the username used to authenticate the client session. This property is used as the default value for the **Connect** method if no username is specified as an argument.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Version Property

Gets a value which returns the current version of the ImapClient class library.

[Visual Basic]

```
Public ReadOnly Property Version As String
```

[C#]

```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the ImapClient class library. This value can be used by an application for validation and debugging purposes.

See Also

ImapClient Class | SocketTools Namespace

ImapClient Methods

The methods of the **ImapClient** class are listed below. For a complete list of **ImapClient** class members, see the ImapClient Members topic.

Public Instance Methods

AttachThread	Attach an instance of the class to the current thread
Cancel	Cancel the current blocking client operation.
CheckMailbox	Create a checkpoint for the currently selected mailbox.
CloseMessage	Closes the current message.
Command	Overloaded. Send a custom command to the mail server.
Connect	Overloaded. Establish a connection with a remote host.
CopyMessage	Copy a message from the current mailbox to another mailbox.
CreateMailbox	Creates a new mailbox on the server.
CreateMessage	Overloaded. Create a new message.
DeleteMailbox	Overloaded. Deletes a mailbox from the server.
DeleteMessage	Overloaded. Flags a message for deletion from the current mailbox.
Disconnect	Terminate the connection with a remote host.
Dispose	Overloaded. Releases all resources used by ImapClient.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
ExamineMailbox	Selects the specified mailbox for read-only access.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetHeader	Overloaded. Returns the value of a header field from the specified message part.
GetHeaders	Overloaded. Retrieves the headers for the specified message from the server.
GetMessage	Overloaded. Retrieve a message from the server and return the contents in a byte array.
GetType (inherited from Object)	Gets the Type of the current instance.
Idle	Overloaded. Enables mailbox status monitoring for the client session.

Initialize	Overloaded. Initialize an instance of the ImapClient class.
OpenMessage	Overloaded. Open the specified message for reading.
Read	Overloaded. Read data from the server and store it in a byte array.
Refresh	Updates the list of available mailboxes.
RenameMailbox	Change the name of a mailbox.
ReselectMailbox	Reselects the current mailbox.
Reset	Reset the internal state of the object, resetting all properties to their default values.
SearchMailbox	Overloaded. Search the current mailbox for messages that match the specified criteria and character set.
SelectMailbox	Selects the specified mailbox for read-write access.
StoreMessage	Overloaded. Retrieve a message from the current mailbox and store it in a file on the local system.
SubscribeMailbox	Overloaded. Subscribes the user to the specified mailbox.
ToString (inherited from Object)	Returns a String that represents the current Object.
UndeleteMessage	Removes the deletion flag for the specified message.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
UnselectMailbox	Overloaded. Unselects the current mailbox.
UnsubscribeMailbox	Overloaded. Unsubscribes the user from the specified mailbox.
Write	Overloaded. Write one or more bytes of data to the server.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the ImapClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.AttachThread Method

Attach an instance of the class to the current thread

[Visual Basic]

```
Public Function AttachThread() As Boolean
```

[C#]

```
public bool AttachThread();
```

Return Value

A boolean value which specifies if the client could be attached to the current thread. If this method returns **false**, the client could not be attached to the thread and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

When an instance of the class is created it is associated with the current thread that created it. Normally, if another thread attempts to perform an operation using that instance, an error is returned. This is used to ensure that other threads cannot interfere with an operation being performed by the owner thread. In some cases, it may be desirable for one thread in an application to create an instance of the class, establish a connection and then pass that instance to another worker thread. The **AttachThread** method can be used to change the ownership of the class instance to the new worker thread.

This method should be called by the new thread immediately after it has been created, and if the new thread does not release the handle itself, the ownership of the handle should be restored by the original thread. Under no circumstances should **AttachThread** be used to forcibly destroy an instance of a class allocated by another thread while a blocking operation is in progress. To cancel a blocking operation, use the **Cancel** method and then delete the class instance after the blocking function exits and control is returned to the current thread.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Cancel Method

Cancel the current blocking client operation.

[Visual Basic]

```
Public Sub Cancel()
```

[C#]

```
public void Cancel();
```

Remarks

When the **Cancel** method is called, the blocking client operation will not immediately fail. An internal flag is set which causes the blocking operation to exit with an error. This means that the application cannot cancel an operation and immediately perform some other blocking function. Instead it must allow the calling stack to unwind, returning back to the blocking operation before making any further function calls.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.CheckMailbox Method

Create a checkpoint for the currently selected mailbox.

[Visual Basic]
```
Public Function CheckMailbox() As Boolean
```

[C#]
```
public bool CheckMailbox();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CheckMailbox** method requests that the server create a checkpoint of the currently selected mailbox, and updates the current number of new, unread messages available to the client.

When the client requests a checkpoint, the server may perform implementation-dependent housekeeping for that mailbox, such updating the mailbox on disk with the current state of the mailbox in memory. On some systems this command has no effect other than to update the client with the current number of messages in the mailbox.

This function actually sends two IMAP commands. The first is the CHECK command, followed by the NOOP command to poll for any new messages that have arrived. In addition to polling the server for new messages, this command can also be used to ensure the idle timer on the server does not expire and force a disconnect from the client.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.CloseMessage Method

Closes the current message.

[Visual Basic]
```
Public Function CloseMessage() As Boolean
```

[C#]
```
public bool CloseMessage();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CloseMessage** method closes the current message. If there is any remaining data left to be read from the message, it will be read and discarded.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Command Method

Send a custom command to the mail server.

Overload List

Send a custom command to the mail server.

public bool Command(string);

Send a custom command to the mail server.

public bool Command(string,string);

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Command Method (String)

Send a custom command to the mail server.

[Visual Basic]

```
Overloads Public Function Command( _
   ByVal command As String _
) As Boolean
```

[C#]

```
public bool Command(
   string command
);
```

Parameters

command

A string which specifies the command to send. Valid commands vary based on the Internet protocol and the type of server that the client is connected to. Consult the protocol standard and/or the technical reference documentation for the server to determine what commands may be issued by a client application.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Command** method sends a command to the remote host and processes the result code sent back in response to that command. This method can be used to send custom commands to a server to take advantage of features or capabilities that may not be supported internally by the class library.

To determine if the command was successful, check the value of the **ResultCode** property. To obtain additional information returned by the server in response to the command, check the value of the **ResultString** property.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Command Overload List

ImapClient.Command Method (String, String)

Send a custom command to the mail server.

```
[Visual Basic]
Overloads Public Function Command( _
   ByVal command As String, _
   ByVal parameter As String _
) As Boolean
```

```
[C#]
public bool Command(
   string command,
   string parameter
);
```

Parameters

command

A string which specifies the command to send. Valid commands vary based on the Internet protocol and the type of server that the client is connected to. Consult the protocol standard and/or the technical reference documentation for the server to determine what commands may be issued by a client application.

parameter

A string which specifies one or more parameters to be sent along with the command. If more than one parameter is required, they should be separated by a single space character. Consult the protocol standard and/or technical reference documentation for the server to determine what parameters should be provided when issuing a specific command. If no parameters are required for the command, this argument may be omitted.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Command** method sends a command to the remote host and processes the result code sent back in response to that command. This method can be used to send custom commands to a server to take advantage of features or capabilities that may not be supported internally by the class library.

To determine if the command was successful, check the value of the **ResultCode** property. To obtain additional information returned by the server in response to the command, check the value of the **ResultString** property.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Command Overload List

ImapClient.Connect Method

Establish a connection with a remote host.

Overload List

Establish a connection with a remote host.

public bool Connect();

Establish a connection with a remote host.

public bool Connect(string,int);

Establish a connection with a remote host.

public bool Connect(string,int,int);

Establish a connection with a remote host.

public bool Connect(string,int,int,ImapOptions);

Establish a connection with a remote host.

public bool Connect(string,int,string,string);

Establish a connection with a remote host.

public bool Connect(string,int,string,string,int);

Establish a connection with a remote host.

public bool Connect(string,int,string,string,int,ImapOptions);

Establish a connection with a remote host.

public bool Connect(string,string,string);

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Connect Method ()

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect() As Boolean
```

[C#]

```
public bool Connect();
```

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection using assigned property values.

The value of the **HostName** or **HostAddress** property will be used to determine the host name or address to connect to.

The value of the **RemotePort** property will be used to determine the port number to connect to.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Connect Overload List

ImapClient.Connect Method (String, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Connect Overload List

ImapClient.Connect Method (String, Int32, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   int timeout
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Connect Overload List

ImapClient.Connect Method (String, Int32, Int32, ImapOptions)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer, _
   ByVal options As ImapOptions _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   int timeout,
   ImapOptions options
);
```

Parameters

hostName
A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort
An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

timeout
An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

options
One or more of the ImapOptions enumeration flags.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Connect Overload List

ImapClient.Connect Method (String, Int32, String, String)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword
);
```

Parameters

hostName
A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort
An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName
A string which specifies a username used to authenticate the client session.

userPassword
A string which specifies the password used to authenticate the client session.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Connect Overload List

ImapClient.Connect Method (String, Int32, String, String, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   int timeout
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName

A string which specifies a username used to authenticate the client session.

userPassword

A string which specifies the password used to authenticate the client session.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Connect Overload List

ImapClient.Connect Method (String, Int32, String, String, Int32, ImapOptions)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal timeout As Integer, _
   ByVal options As ImapOptions _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   int timeout,
   ImapOptions options
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName

A string which specifies a username used to authenticate the client session.

userPassword

A string which specifies the password used to authenticate the client session.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

options

One or more of the ImapOptions enumeration flags.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Connect Overload List

ImapClient.Connect Method (String, String, String)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   string userName,
   string userPassword
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

userName

A string which specifies a username used to authenticate the client session.

userPassword

A string which specifies the password used to authenticate the client session.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Connect Overload List

ImapClient.CopyMessage Method

Copy a message from the current mailbox to another mailbox.

[Visual Basic]

```
Public Function CopyMessage( _
   ByVal messageId As Integer, _
   ByVal mailboxName As String _
) As Boolean
```

[C#]

```
public bool CopyMessage(
   int messageId,
   string mailboxName
);
```

Parameters

messageId

The message identifier which specifies which message will be copied to the mailbox. This value must be greater than zero and specify a valid message number.

mailboxName

A string which specifies the name of the mailbox that the message will be copied to. The mailbox must already exist, and the client must have the appropriate access rights to modify the mailbox.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CopyMessage** method copies a message from the current mailbox to the specified mailbox. The message is appended to the mailbox, and the message flags and internal date are preserved. If the mailbox does not exist, the method will fail. To create a new mailbox, use the **CreateMailbox** method. A message can be copied within the same mailbox, in which case the server may flag it as a new message.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.CreateMailbox Method

Creates a new mailbox on the server.

[Visual Basic]

```
Public Function CreateMailbox( _
   ByVal mailboxName As String _
) As Boolean
```

[C#]

```
public bool CreateMailbox(
   string mailboxName
);
```

Parameters

mailboxName

A string which specifies the name of the new mailbox to be created.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateMailbox** method creates a new mailbox on the server. If the mailbox name is suffixed with the server's hierarchy delimiter, this indicates to the server that the client intends to create mailbox names under the specified name in the hierarchy. If superior hierarchical names are specified in the mailbox name, then the server may automatically create them as needed. For example, if the mailbox name "Mail/Office/Projects" is specified and "Mail/Office" does not exist, it may be automatically created by the server.

The special mailbox name INBOX is reserved, and cannot be created. It is recommended that mailbox names only consist of printable ASCII characters, and the special characters "*" and "%" should be avoided.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.CreateMessage Method

Create a new message.

Overload List

Create a new message.

public bool CreateMessage(byte[],int);

Create a new message.

public bool CreateMessage(byte[],int,ImapFlags);

Create a new message.

public bool CreateMessage(string);

Create a new message.

public bool CreateMessage(string,ImapFlags);

Create a new message.

public bool CreateMessage(string,byte[],int,ImapFlags);

Create a new message.

public bool CreateMessage(string,string,ImapFlags);

See Also

ImapClient Class | SocketTools Namespace

ImapClient.CreateMessage Method (Byte[], Int32)

Create a new message.

[Visual Basic]

```
Overloads Public Function CreateMessage( _
   ByVal message As Byte(), _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool CreateMessage(
   byte[] message,
   int length
);
```

Parameters

message

A byte array that contains the message data.

length

An integer value which specifies the size of the message in bytes. This value cannot be larger than the size of the message buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateMessage** method creates a new message, appending it to the contents of the specified mailbox. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The message will be created in the mailbox specified by the **MailboxName** property.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.CreateMessage Overload List

ImapClient.CreateMessage Method (Byte[], Int32, ImapFlags)

Create a new message.

[Visual Basic]

```
Overloads Public Function CreateMessage( _
   ByVal message As Byte(), _
   ByVal length As Integer, _
   ByVal messageFlags As ImapFlags _
) As Boolean
```

[C#]

```
public bool CreateMessage(
   byte[] message,
   int length,
   ImapFlags messageFlags
);
```

Parameters

message

A byte array that contains the message data.

length

An integer value which specifies the size of the message in bytes. This value cannot be larger than the size of the message buffer specified by the caller.

messageFlags

An ImapFlags enumeration value which specifies one or more message flags. One or more of the following flags may be used:

Flag	Description
imapFlagNone	The message will be created with no flags set.
imapFlagAnswered	The message has been answered.
imapFlagDraft	The message is a draft copy.
imapFlagUrgent	The message is urgent.
imapFlagSeen	The message has already been read.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateMessage** method creates a new message, appending it to the contents of the specified mailbox. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The message will be created in the mailbox specified by the **MailboxName** property.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.CreateMessage Overload List

ImapClient.CreateMessage Method (String)

Create a new message.

[Visual Basic]

```
Overloads Public Function CreateMessage( _
   ByVal message As String _
) As Boolean
```

[C#]

```
public bool CreateMessage(
   string message
);
```

Parameters

message

A string that contains the message data.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateMessage** method creates a new message, appending it to the contents of the specified mailbox. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The message will be created in the mailbox specified by the **MailboxName** property.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.CreateMessage Overload List

ImapClient.CreateMessage Method (String, ImapFlags)

Create a new message.

```
[Visual Basic]
Overloads Public Function CreateMessage( _
   ByVal message As String, _
   ByVal messageFlags As ImapFlags _
) As Boolean
```

```
[C#]
public bool CreateMessage(
   string message,
   ImapFlags messageFlags
);
```

Parameters

message

A string that contains the message data.

messageFlags

An ImapFlags enumeration value which specifies one or more message flags. One or more of the following flags may be used:

Flag	Description
imapFlagNone	The message will be created with no flags set.
imapFlagAnswered	The message has been answered.
imapFlagDraft	The message is a draft copy.
imapFlagUrgent	The message is urgent.
imapFlagSeen	The message has already been read.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateMessage** method creates a new message, appending it to the contents of the specified mailbox. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The message will be created in the mailbox specified by the **MailboxName** property.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.CreateMessage Overload List

ImapClient.CreateMessage Method (String, Byte[], Int32, ImapFlags)

Create a new message.

[Visual Basic]

```
Overloads Public Function CreateMessage( _
   ByVal mailboxName As String, _
   ByVal message As Byte(), _
   ByVal length As Integer, _
   ByVal messageFlags As ImapFlags _
) As Boolean
```

[C#]

```
public bool CreateMessage(
   string mailboxName,
   byte[] message,
   int length,
   ImapFlags messageFlags
);
```

Parameters

mailboxName
A string which specifies the name of the mailbox the message will be created in.

message
A byte array that contains the message data.

length
An integer value which specifies the size of the message in bytes. This value cannot be larger than the size of the message buffer specified by the caller.

messageFlags

An ImapFlags enumeration value which specifies one or more message flags. One or more of the following flags may be used:

Flag	Description
imapFlagNone	The message will be created with no flags set.
imapFlagAnswered	The message has been answered.
imapFlagDraft	The message is a draft copy.
imapFlagUrgent	The message is urgent.
imapFlagSeen	The message has already been read.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateMessage** method creates a new message, appending it to the contents of the specified mailbox. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.CreateMessage Overload List

ImapClient.CreateMessage Method (String, String, ImapFlags)

Create a new message.

[Visual Basic]

```
Overloads Public Function CreateMessage( _
   ByVal mailboxName As String, _
   ByVal message As String, _
   ByVal messageFlags As ImapFlags _
) As Boolean
```

[C#]

```
public bool CreateMessage(
   string mailboxName,
   string message,
   ImapFlags messageFlags
);
```

Parameters

mailboxName

A string which specifies the name of the mailbox the message will be created in.

message

A string that contains the message data.

messageFlags

An ImapFlags enumeration value which specifies one or more message flags. One or more of the following flags may be used:

Flag	Description
imapFlagNone	The message will be created with no flags set.
imapFlagAnswered	The message has been answered.
imapFlagDraft	The message is a draft copy.
imapFlagUrgent	The message is urgent.
imapFlagSeen	The message has already been read.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateMessage** method creates a new message, appending it to the contents of the specified mailbox. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.CreateMessage Overload List

ImapClient.DeleteMailbox Method

Deletes the currently selected mailbox from the server.

Overload List

Deletes the currently selected mailbox from the server.

public bool DeleteMailbox();

Deletes a mailbox from the server.

public bool DeleteMailbox(string);

See Also

ImapClient Class | SocketTools Namespace

ImapClient.DeleteMailbox Method ()

Deletes the currently selected mailbox from the server.

[Visual Basic]

```
Overloads Public Function DeleteMailbox() As Boolean
```

[C#]

```
public bool DeleteMailbox();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **DeleteMailbox** method deletes the currently selected mailbox from the server. The current mailbox will be automatically unselected and any messages marked for deletion will be expunged before the mailbox is removed. If the delete operation fails, the client will remain in an unselected state until either the **ExamineMailbox** or **SelectMailbox** method is called

A mailbox cannot be deleted if it contains inferior hierarchical names and has the **imapFlagNoSelect** attribute. On most systems this is the case when the mailbox name references a directory on the server, and that directory contains other subdirectories or mailboxes. To remove the current mailbox, you must first delete any child mailboxes that exist.

The special mailbox named INBOX cannot be deleted.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.DeleteMailbox Overload List

ImapClient.DeleteMailbox Method (String)

Deletes a mailbox from the server.

[Visual Basic]

```
Overloads Public Function DeleteMailbox( _
   ByVal mailboxName As String _
) As Boolean
```

[C#]

```
public bool DeleteMailbox(
   string mailboxName
);
```

Parameters

mailboxName

A string which specifies the name of the new mailbox to be deleted.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **DeleteMailbox** method deletes a mailbox from the server. A mailbox cannot be deleted if it contains inferior hierarchical names and has the **imapFlagNoSelect** attribute. On most systems this is the case when the mailbox name references a directory on the server, and that directory contains other subdirectories or mailboxes. To remove the mailbox, you must first delete any child mailboxes that exist.

If the mailbox that is deleted is the currently selected mailbox, it will be automatically unselected and any messages marked for deletion will be expunged before the mailbox is removed. If the delete operation fails, the client will remain in an unselected state until either the **ExamineMailbox** or **SelectMailbox** method is called.

The special mailbox named INBOX cannot be deleted.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.DeleteMailbox Overload List

ImapClient.DeleteMessage Method

Flags the current message for deletion from the mailbox.

Overload List

Flags the current message for deletion from the mailbox.

public bool DeleteMessage();

Flags a message for deletion from the current mailbox.

public bool DeleteMessage(int);

See Also

ImapClient Class | SocketTools Namespace

ImapClient.DeleteMessage Method ()

Flags the current message for deletion from the mailbox.

[Visual Basic]

```
Overloads Public Function DeleteMessage() As Boolean
```

[C#]

```
public bool DeleteMessage();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method only flags the current message for deletion, it is not actually deleted until the mailbox is expunged or another mailbox is selected. This function will return an error if the current mailbox is in read-only mode, such as if it was selected using the **ExamineMailbox** method.

It is important to note that unlike the POP3 protocol, a message that is flagged for deletion is still accessible on the IMAP server until the mailbox is expunged. This means, for example, that a deleted message can still be retrieved using the **GetMessage** method.

To determine if a message has been flagged for deletion, set the **Message** property to the message number and then check the value of the **MessageFlags** property to determine if the **ImapFlags.flagDeleted** flag has been set.

To remove the deletion flag from the message, use the **UndeleteMessage** method. To prevent all messages in the current mailbox from being expunged, use the **ReselectMailbox** function to reset the current mailbox state. Calling the **Reset** method will also unselect the current mailbox without expunging deleted messages.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.DeleteMessage Overload List

ImapClient.DeleteMessage Method (Int32)

Flags a message for deletion from the current mailbox.

[Visual Basic]

```
Overloads Public Function DeleteMessage( _
   ByVal messageId As Integer _
) As Boolean
```

[C#]

```
public bool DeleteMessage(
   int messageId
);
```

Parameters

messageId

Number of message to delete from the server. This value must be greater than zero. The first message in the mailbox is message number one.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method only flags the message for deletion, it is not actually deleted until the mailbox is expunged or another mailbox is selected. This function will return an error if the current mailbox is in read-only mode, such as if it was selected using the **ExamineMailbox** method.

It is important to note that unlike the POP3 protocol, a message that is flagged for deletion is still accessible on the IMAP server until the mailbox is expunged. This means, for example, that a deleted message can still be retrieved using the **GetMessage** method.

To determine if a message has been flagged for deletion, set the **Message** property to the message number and then check the value of the **MessageFlags** property to determine if the **ImapFlags.flagDeleted** flag has been set.

To remove the deletion flag from the message, use the **UndeleteMessage** method. To prevent all messages in the current mailbox from being expunged, use the **ReselectMailbox** function to reset the current mailbox state. Calling the **Reset** method will also unselect the current mailbox without expunging deleted messages.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.DeleteMessage Overload List

ImapClient.Disconnect Method

Terminate the connection with a remote host.

[Visual Basic]

```
Public Sub Disconnect()
```

[C#]

```
public void Disconnect();
```

Remarks

The **Disconnect** method terminates the connection with the remote host and releases the client handle allocated by the class. Note that the socket is not immediately released when the connection is terminated and will enter a wait state for two minutes. After the time wait period has elapsed, the client will be released by the operating system. This is a normal safety mechanism to handle any packets that may arrive after the connection has been closed.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Dispose Method

Releases all resources used by ImapClient.

Overload List

Releases all resources used by ImapClient.

public void Dispose();

Releases the unmanaged resources allocated by the ImapClient class and optionally releases the managed resources.

protected virtual void Dispose(bool);

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Dispose Method ()

Releases all resources used by ImapClient.

[Visual Basic]

```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]

```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Dispose Overload List

ImapClient.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the ImapClient class and optionally releases the managed resources.

[Visual Basic]

```
Overridable Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]

```
protected virtual void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **ImapClient** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Dispose Overload List

ImapClient.ExamineMailbox Method

Selects the specified mailbox for read-only access.

[Visual Basic]

```
Public Function ExamineMailbox( _
   ByVal mailboxName As String _
) As Boolean
```

[C#]

```
public bool ExamineMailbox(
   string mailboxName
);
```

Parameters

mailboxName
A string that specifies the name of the mailbox to be examined.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ExamineMailbox** method is used to select a mailbox in read-only mode. Messages can be read, but they cannot be modified or deleted from the mailbox and new messages will not lose their status as new messages if they are accessed.

If the client has a different mailbox currently selected, that mailbox will be closed and any messages marked for deletion will be expunged. To prevent deleted messages from being removed from the previous mailbox, use the **UnselectMailbox** method prior to examining the new mailbox.

The special case-insensitive mailbox name INBOX is used for new messages. Other mailbox names may or may not be case-sensitive depending on the IMAP server's operating system and implementation.

To access a mailbox in read-write mode, use the **SelectMailbox** method.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Finalize Method

Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.

[Visual Basic]

```
Overrides Protected Sub Finalize()
```

[C#]

```
protected override void Finalize();
```

See Also

ImapClient Class | SocketTools Namespace

ImapClient.GetHeader Method

Returns the value of a header field from the specified message part.

Overload List

Returns the value of a header field from the specified message part.

public bool GetHeader(int,int,string,ref string);

Returns the value of a header field from the specified message.

public bool GetHeader(int,string,ref string);

Returns the value of a header field from the current message.

public bool GetHeader(string,ref string);

See Also

ImapClient Class | SocketTools Namespace

ImapClient.GetHeader Method (Int32, Int32, String, String)

Returns the value of a header field from the specified message part.

[Visual Basic]

```
Overloads Public Function GetHeader( _
   ByVal messageId As Integer, _
   ByVal messagePart As Integer, _
   ByVal headerName As String, _
   ByRef headerValue As String _
) As Boolean
```

[C#]

```
public bool GetHeader(
   int messageId,
   int messagePart,
   string headerName,
   ref string headerValue
);
```

Parameters

messageId
An integer value that specifies the message to retrieve the header value from. This value must be greater than zero. The first message in the mailbox is message number one.

messagePart
An integer value that specifies the message part that the header value is to be be retrieved from. Message parts start with a value of one. A value of zero specifies that the RFC822 header field for the message will be used.

headerName
A string which specifies the message header to retrieve.

headerValue
A string passed by reference which will contain the value of the specified message header if the method is successful.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeader** method returns the value of a header field from the specified message part. This allows an application to be able to easily determine the value of a header such as the sender, or the subject of the message. Any header field, including non-standard extensions, may be returned by this method.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.GetHeader Overload List

ImapClient.GetHeader Method (Int32, String, String)

Returns the value of a header field from the specified message.

[Visual Basic]

```
Overloads Public Function GetHeader( _
   ByVal messageId As Integer, _
   ByVal headerName As String, _
   ByRef headerValue As String _
) As Boolean
```

[C#]

```
public bool GetHeader(
   int messageId,
   string headerName,
   ref string headerValue
);
```

Parameters

messageId
An integer value that specifies the message to retrieve the header value from. This value must be greater than zero. The first message in the mailbox is message number one.

headerName
A string which specifies the message header to retrieve.

headerValue
A string passed by reference which will contain the value of the specified message header if the method is successful.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeader** method returns the value of a header field from the specified message. This allows an application to be able to easily determine the value of a header such as the sender, or the subject of the message. Any header field, including non-standard extensions, may be returned by this method.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.GetHeader Overload List

ImapClient.GetHeader Method (String, String)

Returns the value of a header field from the current message.

[Visual Basic]

```
Overloads Public Function GetHeader( _
   ByVal headerName As String, _
   ByRef headerValue As String _
) As Boolean
```

[C#]

```
public bool GetHeader(
   string headerName,
   ref string headerValue
);
```

Parameters

headerName
A string which specifies the message header to retrieve.

headerValue
A string passed by reference which will contain the value of the specified message header if the method is successful.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeader** method returns the value of a header field from the current message. This allows an application to be able to easily determine the value of a header such as the sender, or the subject of the message. Any header field, including non-standard extensions, may be returned by this method.

The current message number is specified by the value of the **Message** property.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.GetHeader Overload List

ImapClient.GetHeaders Method

Retrieves the headers for the current message from the server.

Overload List

Retrieves the headers for the current message from the server.

public bool GetHeaders(byte[],ref int);

Retrieves the headers for the specified message from the server.

public bool GetHeaders(int,byte[],ref int);

Retrieves the headers for the specified message from the server.

public bool GetHeaders(int,ref string);

Retrieves the headers for the current message from the server.

public bool GetHeaders(ref string);

See Also

ImapClient Class | SocketTools Namespace

ImapClient.GetHeaders Method (Byte[], Int32)

Retrieves the headers for the current message from the server.

[Visual Basic]

```
Overloads Public Function GetHeaders( _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool GetHeaders(
   byte[] buffer,
   ref int length
);
```

Parameters

buffer
A byte array that will contain the message data when the method returns.

length
An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeaders** method is used to retrieve the message headers from the server and copy it into a local buffer. This method will cause the current thread to block until the article transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Note that the header data will be from the first part of the message, not from any additional sections of a multipart message. In other words, the headers such as From, To, Subject and Date will be returned in the buffer. To retrieve the headers from a specific section of a multipart message, you can use the **GetMessage** method and specify the **ImapSections.sectionHeader** option.

The current message number is specified by the value of the **Message** property.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.GetHeaders Overload List

ImapClient.GetHeaders Method (Int32, Byte[], Int32)

Retrieves the headers for the specified message from the server.

[Visual Basic]

```
Overloads Public Function GetHeaders( _
   ByVal messageId As Integer, _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool GetHeaders(
   int messageId,
   byte[] buffer,
   ref int length
);
```

Parameters

messageId

Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

buffer

A byte array that will contain the message data when the method returns.

length

An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeaders** method is used to retrieve the message headers from the server and copy it into a local buffer. This method will cause the current thread to block until the article transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Note that the header data will be from the first part of the message, not from any additional sections of a multipart message. In other words, the headers such as From, To, Subject and Date will be returned in the buffer. To retrieve the headers from a specific section of a multipart message, you can use the **GetMessage** method and specify the **ImapSections.sectionHeader** option.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.GetHeaders Overload List

ImapClient.GetHeaders Method (Int32, String)

Retrieves the headers for the specified message from the server.

[Visual Basic]

```
Overloads Public Function GetHeaders( _
   ByVal messageId As Integer, _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool GetHeaders(
   int messageId,
   ref string buffer
);
```

Parameters

messageId
Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

buffer
A string passed by reference that will contain the message data when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeaders** method is used to retrieve the message headers from the server and copy it into a local buffer. This method will cause the current thread to block until the article transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Note that the header data will be from the first part of the message, not from any additional sections of a multipart message. In other words, the headers such as From, To, Subject and Date will be returned in the buffer. To retrieve the headers from a specific section of a multipart message, you can use the **GetMessage** method and specify the **ImapSections.sectionHeader** option.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.GetHeaders Overload List

ImapClient.GetHeaders Method (String)

Retrieves the headers for the current message from the server.

[Visual Basic]

```
Overloads Public Function GetHeaders( _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool GetHeaders(
   ref string buffer
);
```

Parameters

buffer

A string passed by reference that will contain the message data when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeaders** method is used to retrieve the message headers from the server and copy it into a local buffer. This method will cause the current thread to block until the article transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Note that the header data will be from the first part of the message, not from any additional sections of a multipart message. In other words, the headers such as From, To, Subject and Date will be returned in the buffer. To retrieve the headers from a specific section of a multipart message, you can use the **GetMessage** method and specify the **ImapSections.sectionHeader** option.

The current message number is specified by the value of the **Message** property.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.GetHeaders Overload List

ImapClient.GetMessage Method

Retrieve the current message from the server and return the contents in a byte array.

Overload List

Retrieve the current message from the server and return the contents in a byte array.

public bool GetMessage(byte[],ref int);

Retrieve the current message from the server and return the contents in a byte array.

public bool GetMessage(byte[],ref int,ImapSections);

Retrieve a message from the server and return the contents in a byte array.

public bool GetMessage(int,byte[],ref int);

Retrieve a message from the server and return the contents in a byte array.

public bool GetMessage(int,byte[],ref int,ImapSections);

Retrieve a message from the server and return the contents in a byte array.

public bool GetMessage(int,int,byte[],ref int);

Retrieve a message from the server and return the contents in a byte array.

public bool GetMessage(int,int,byte[],ref int,ImapSections);

Retrieve a message from the server and return the contents in a string.

public bool GetMessage(int,int,ref string);

Retrieve a message from the server and return the contents in a string.

public bool GetMessage(int,int,ref string,ImapSections);

Retrieve a message from the server and return the contents in a string.

public bool GetMessage(int,ref string);

Retrieve a message from the server and return the contents in a string.

public bool GetMessage(int,ref string,ImapSections);

Retrieve the current message from the server and return the contents in a string.

public bool GetMessage(ref string);

Retrieve the current message message from the server and return the contents in a string.

public bool GetMessage(ref string,ImapSections);

See Also

ImapClient Class | SocketTools Namespace

ImapClient.GetMessage Method (Byte[], Int32)

Retrieve the current message from the server and return the contents in a byte array.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool GetMessage(
   byte[] buffer,
   ref int length
);
```

Parameters

buffer

A byte array that the message data will be stored in.

length

An integer value passed by reference which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller. When the method returns, this parameter will be updated with the actual number of bytes copied into the array.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve the current message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The current message number is specified by the value of the **Message** property.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.GetMessage Overload List

ImapClient.GetMessage Method (Byte[], Int32, ImapSections)

Retrieve the current message from the server and return the contents in a byte array.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByVal buffer As Byte(), _
   ByRef length As Integer, _
   ByVal options As ImapSections _
) As Boolean
```

[C#]

```
public bool GetMessage(
   byte[] buffer,
   ref int length,
   ImapSections options
);
```

Parameters

buffer

A byte array that the message data will be stored in.

length

An integer value passed by reference which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller. When the method returns, this parameter will be updated with the actual number of bytes copied into the array.

options

An ImapSections enumeration value which specifies which section of the message to return.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve the current message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The current message number is specified by the value of the **Message** property.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.GetMessage Overload List

ImapClient.GetMessage Method (Int32, Byte[], Int32)

Retrieve a message from the server and return the contents in a byte array.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByVal messageId As Integer, _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool GetMessage(
   int messageId,
   byte[] buffer,
   ref int length
);
```

Parameters

messageId

Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

buffer

A byte array that the message data will be stored in.

length

An integer value passed by reference which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller. When the method returns, this parameter will be updated with the actual number of bytes copied into the array.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve a message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.GetMessage Overload List

ImapClient.GetMessage Method (Int32, Byte[], Int32, ImapSections)

Retrieve a message from the server and return the contents in a byte array.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByVal messageId As Integer, _
   ByVal buffer As Byte(), _
   ByRef length As Integer, _
   ByVal options As ImapSections _
) As Boolean
```

[C#]

```
public bool GetMessage(
   int messageId,
   byte[] buffer,
   ref int length,
   ImapSections options
);
```

Parameters

messageId

Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

buffer

A byte array that the message data will be stored in.

length

An integer value passed by reference which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller. When the method returns, this parameter will be updated with the actual number of bytes copied into the array.

options

An ImapSections enumeration value which specifies which section of the message to return.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve a message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.GetMessage Overload List

ImapClient.GetMessage Method (Int32, Int32, Byte[], Int32)

Retrieve a message from the server and return the contents in a byte array.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByVal messageId As Integer, _
   ByVal messagePart As Integer, _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool GetMessage(
   int messageId,
   int messagePart,
   byte[] buffer,
   ref int length
);
```

Parameters

messageId

Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

messagePart

An integer value that specifies the message part that should be retrieved. A value of zero specifies that the complete message should be returned. If the message is a multipart MIME message, message parts start with a value of one.

buffer

A byte array that the message data will be stored in.

length

An integer value passed by reference which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller. When the method returns, this parameter will be updated with the actual number of bytes copied into the array.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve a message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.GetMessage Overload List

ImapClient.GetMessage Method (Int32, Int32, Byte[], Int32, ImapSections)

Retrieve a message from the server and return the contents in a byte array.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByVal messageId As Integer, _
   ByVal messagePart As Integer, _
   ByVal buffer As Byte(), _
   ByRef length As Integer, _
   ByVal options As ImapSections _
) As Boolean
```

[C#]

```
public bool GetMessage(
   int messageId,
   int messagePart,
   byte[] buffer,
   ref int length,
   ImapSections options
);
```

Parameters

messageId
Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

messagePart
An integer value that specifies the message part that should be retrieved. A value of zero specifies that the complete message should be returned. If the message is a multipart MIME message, message parts start with a value of one.

buffer
A byte array that the message data will be stored in.

length
An integer value passed by reference which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller. When the method returns, this parameter will be updated with the actual number of bytes copied into the array.

options
An ImapSections enumeration value which specifies which section of the message to return.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve a message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.GetMessage Overload List

ImapClient.GetMessage Method (Int32, Int32, String)

Retrieve a message from the server and return the contents in a string.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByVal messageId As Integer, _
   ByVal messagePart As Integer, _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool GetMessage(
   int messageId,
   int messagePart,
   ref string buffer
);
```

Parameters

messageId

Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

messagePart

An integer value that specifies the message part that should be retrieved. A value of zero specifies that the complete message should be returned. If the message is a multipart MIME message, message parts start with a value of one.

buffer

A string passed by reference that will contain the message data when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve a message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.GetMessage Overload List

ImapClient.GetMessage Method (Int32, Int32, String, ImapSections)

Retrieve a message from the server and return the contents in a string.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByVal messageId As Integer, _
   ByVal messagePart As Integer, _
   ByRef buffer As String, _
   ByVal options As ImapSections _
) As Boolean
```

[C#]

```
public bool GetMessage(
   int messageId,
   int messagePart,
   ref string buffer,
   ImapSections options
);
```

Parameters

messageId
Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

messagePart
An integer value that specifies the message part that should be retrieved. A value of zero specifies that the complete message should be returned. If the message is a multipart MIME message, message parts start with a value of one.

buffer
A string passed by reference that will contain the message data when the method returns.

options
An ImapSections enumeration value which specifies which section of the message to return.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve a message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.GetMessage Overload List

ImapClient.GetMessage Method (Int32, String)

Retrieve a message from the server and return the contents in a string.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByVal messageId As Integer, _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool GetMessage(
   int messageId,
   ref string buffer
);
```

Parameters

messageId

Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

buffer

A string passed by reference that will contain the message data when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve a message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.GetMessage Overload List

ImapClient.GetMessage Method (Int32, String, ImapSections)

Retrieve a message from the server and return the contents in a string.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByVal messageId As Integer, _
   ByRef buffer As String, _
   ByVal options As ImapSections _
) As Boolean
```

[C#]

```
public bool GetMessage(
   int messageId,
   ref string buffer,
   ImapSections options
);
```

Parameters

messageId

Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

buffer

A string passed by reference that will contain the message data when the method returns.

options

An ImapSections enumeration value which specifies which section of the message to return.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve a message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.GetMessage Overload List

ImapClient.GetMessage Method (String)

Retrieve the current message from the server and return the contents in a string.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool GetMessage(
   ref string buffer
);
```

Parameters

buffer
A string passed by reference that will contain the message data when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve the current message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The current message number is specified by the value of the **Message** property.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.GetMessage Overload List

ImapClient.GetMessage Method (String, ImapSections)

Retrieve the current message message from the server and return the contents in a string.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByRef buffer As String, _
   ByVal options As ImapSections _
) As Boolean
```

[C#]

```
public bool GetMessage(
   ref string buffer,
   ImapSections options
);
```

Parameters

buffer

A string passed by reference that will contain the message data when the method returns.

options

An ImapSections enumeration value which specifies which section of the message to return.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve the current message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The current message number is specified by the value of the **Message** property.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.GetMessage Overload List

ImapClient.Idle Method

Enables mailbox status monitoring for the client session.

Overload List

Enables mailbox status monitoring for the client session.

public bool Idle();

Enables mailbox status monitoring for the client session.

public bool Idle(IdleOptions,int);

See Also

ImapClient Class | SocketTools Namespace | OnUpdate

ImapClient.Idle Method ()

Enables mailbox status monitoring for the client session.

[Visual Basic]

```
Overloads Public Function Idle() As Boolean
```

[C#]

```
public bool Idle();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

Many IMAP servers support the ability to asynchronously send status updates to the client, rather than have the client periodically poll the server. The client enables this feature by calling the **Idle** method and providing an event handler for the **OnUpdate** event. Typically these events inform the client that a new message has arrived or that a message has been expunged from the mailbox.

The **Idle** method begins monitoring the client session asynchronously and returns control immediately to the caller. If the server sends a update notification to the client, the **OnUpdate** event will fire with information about the status change. Sending an IMAP command to the server will cause the client to stop monitoring the session for status changes. To explicitly stop monitoring the session, use the **Cancel** method.

This method works by sending the IDLE command to the server and starting a worker thread which monitors the connection and looks for untagged responses issued by the server. Events will be generated for EXISTS, EXPUNGE and RECENT messages. Note that some servers may periodically send untagged OK messages to the client, indicating that the connection is still active; these messages are explicitly ignored.

The **OnUpdate** event is invoked within the context of the worker thread that is monitoring the client session. Because of this, applications should not directly update the user interface from within the event handler. For example, if the server sends a notification that a new email message has arrived, the application should not attempt to read the new message and update the user interface directly from within the event handler. Instead, it should create a delegate and use the **Control.Invoke** method to marshal the call to the thread that owns the control's window handle. Failure to do this can cause the application to become unstable. For more information, refer to the **Control.Invoke** method in the .NET Framework documentation.

An application should never make an assumption about how a particular server may send update notifications to the client. Servers can be configured to use different intervals at which notifications are sent. For example, a server may send new message notifications immediately, but may periodically notify the client when a message has been expunged. Alternatively, a server may only send notifications at fixed intervals, in which case the client would not be notified of any new messages until the interval period is reached. It is not possible for a client to know what a particular server's update interval is. Applications that require that degree of control should not use the **Idle** method and should poll the server instead.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Idle Overload List | OnUpdate

ImapClient.Idle Method (IdleOptions, Int32)

Enables mailbox status monitoring for the client session.

[Visual Basic]

```
Overloads Public Function Idle( _
   ByVal options As IdleOptions, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Idle(
   IdleOptions options,
   int timeout
);
```

Parameters

options

One or more of the IdleOptions enumeration flags.

timeout

Specifies the timeout period in seconds to wait for a notification from the server. This parameter is only used when the **ImapIdle.idleWait** option has been specified.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

Many IMAP servers support the ability to asynchronously send status updates to the client, rather than have the client periodically poll the server. The client enables this feature by calling the **Idle** method and providing an event handler for the **OnUpdate** event. Typically these events inform the client that a new message has arrived or that a message has been expunged from the mailbox.

The **Idle** method can operate in one of two modes, based on the options specified by the caller. If the option **idleNoWait** is specified, the method begins monitoring the client session asynchronously and returns control immediately to the caller. If the server sends a update notification to the client, the **OnUpdate** event will fire with information about the status change. If the option **idleWait** is specified, the method will block waiting for the server to send a notification message to the client. The method will return when either a message is received or the timeout period is exceeded. Sending an IMAP command to the server will cause the client to stop monitoring the session for status changes. To explicitly stop monitoring the session, use the **Cancel** method.

This method works by sending the IDLE command to the server and starting a worker thread which monitors the connection and looks for untagged responses issued by the server. Events will be generated for EXISTS, EXPUNGE and RECENT messages. Note that some servers may periodically send untagged OK messages to the client, indicating that the connection is still active; these messages are explicitly ignored.

The **OnUpdate** event is invoked within the context of the worker thread that is monitoring the client session. Because of this, applications should not directly update the user interface from within the event handler. For example, if the server sends a notification that a new email message has arrived, the application should not attempt to read the new message and update the user interface directly from within the event handler. Instead, it should create a delegate and use the **Control.Invoke** method to marshal the call to the thread that owns the control's window handle. Failure to do this can cause the application to become unstable. For more information, refer to the **Control.Invoke** method in the .NET Framework documentation.

An application should never make an assumption about how a particular server may send update notifications to the client. Servers can be configured to use different intervals at which notifications are sent. For example, a server may send new message notifications immediately, but may periodically notify the client when a message has been expunged. Alternatively, a server may only send notifications at fixed intervals, in which case the client would not be notified of any new messages until the interval period is reached. It is not possible for a client to know what a particular server's update interval is. Applications that require that degree of control should not use the **Idle** method and should poll the server instead.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Idle Overload List | OnUpdate

ImapClient.Initialize Method

Initialize an instance of the ImapClient class.

Overload List

Initialize an instance of the ImapClient class.

public bool Initialize();

Initialize an instance of the ImapClient class.

public bool Initialize(string);

See Also

ImapClient Class | SocketTools Namespace | Uninitialize Method

ImapClient.Initialize Method ()

Initialize an instance of the ImapClient class.

[Visual Basic]
```
Overloads Public Function Initialize() As Boolean
```

[C#]
```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the ImapClient class, loading the networking library and allocating resources for the current thread. Typically it is not necessary to explicitly call this method because the instance of the class is initialized by the class constructor. However, if the **Uninitialize** method is called, the class must be re-initialized before any other methods are called.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Initialize Overload List | Uninitialize Method

ImapClient.Initialize Method (String)

Initialize an instance of the ImapClient class.

[Visual Basic]
```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]
```
public bool Initialize(
   string licenseKey
);
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the ImapClient class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the ImapClient class.

Example

The following example shows how to use the Initialize method to initialize an instance of the class. This example assumes that the license key string has been defined in code.

```
SocketTools.ImapClient imapClient = new SocketTools.ImapClient();

if (imapClient.Initialize(strLicenseKey) == false)
{
    MessageBox.Show(imapClient.LastErrorString, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    return;
}
```

```
Dim imapClient As New SocketTools.ImapClient

If imapClient.Initialize(strLicenseKey) = False Then
    MsgBox(imapClient.LastErrorString, vbIconExclamation)
    Exit Sub
End If
```

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

ImapClient.OpenMessage Method

Open the current message for reading.

Overload List

Open the current message for reading.

public bool OpenMessage();

Open the specified message for reading.

public bool OpenMessage(int);

Open the specified message for reading.

public bool OpenMessage(int,int);

Open the specified message for reading.

public bool OpenMessage(int,int,ImapSections);

Open the specified message for reading.

public bool OpenMessage(int,int,int,ref int,ImapSections);

Open the specified message for reading.

public bool OpenMessage(int,int,ref int,ImapSections);

See Also

ImapClient Class | SocketTools Namespace

ImapClient.OpenMessage Method ()

Open the current message for reading.

[Visual Basic]

```
Overloads Public Function OpenMessage() As Boolean
```

[C#]

```
public bool OpenMessage();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **OpenMessage** method opens the current message in the mailbox. The client can then read the contents of the message using the **Read** method, and once all of the data has been read, the message should be closed by calling the **CloseMessage** method.

The current message number is specified by the value of the **Message** property.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.OpenMessage Overload List

ImapClient.OpenMessage Method (Int32)

Open the specified message for reading.

[Visual Basic]

```
Overloads Public Function OpenMessage( _
   ByVal messageId As Integer _
) As Boolean
```

[C#]

```
public bool OpenMessage(
   int messageId
);
```

Parameters

messageId

Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **OpenMessage** method opens a message in the current mailbox. The client can then read the contents of the message using the **Read** method, and once all of the data has been read, the message should be closed by calling the **CloseMessage** method.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.OpenMessage Overload List

ImapClient.OpenMessage Method (Int32, Int32)

Open the specified message for reading.

[Visual Basic]

```
Overloads Public Function OpenMessage( _
   ByVal messageId As Integer, _
   ByVal messagePart As Integer _
) As Boolean
```

[C#]

```
public bool OpenMessage(
   int messageId,
   int messagePart
);
```

Parameters

messageId

Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

messagePart

An integer value that specifies the message part that should be retrieved. A value of zero specifies that the complete message should be returned. If the message is a multipart MIME message, message parts start with a value of one.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **OpenMessage** method opens a message or a specific part of a multipart message in the current mailbox. The client can then read the contents of the message using the **Read** method, and once all of the data has been read, the message should be closed by calling the **CloseMessage** method.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.OpenMessage Overload List

ImapClient.OpenMessage Method (Int32, Int32, ImapSections)

Open the specified message for reading.

[Visual Basic]

```
Overloads Public Function OpenMessage( _
   ByVal messageId As Integer, _
   ByVal messagePart As Integer, _
   ByVal options As ImapSections _
) As Boolean
```

[C#]

```
public bool OpenMessage(
   int messageId,
   int messagePart,
   ImapSections options
);
```

Parameters

messageId
Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

messagePart
An integer value that specifies the message part that should be retrieved. A value of zero specifies that the complete message should be returned. If the message is a multipart MIME message, message parts start with a value of one.

options
An ImapSections enumeration value which specifies which section of the message to return.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **OpenMessage** method opens a message in the current mailbox. The client can then read the contents of the message using the **Read** method, and once all of the data has been read, the message should be closed by calling the **CloseMessage** method.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.OpenMessage Overload List

ImapClient.OpenMessage Method (Int32, Int32, Int32, Int32, ImapSections)

Open the specified message for reading.

[Visual Basic]

```
Overloads Public Function OpenMessage( _
   ByVal messageId As Integer, _
   ByVal messagePart As Integer, _
   ByVal offset As Integer, _
   ByRef length As Integer, _
   ByVal options As ImapSections _
) As Boolean
```

[C#]

```
public bool OpenMessage(
   int messageId,
   int messagePart,
   int offset,
   ref int length,
   ImapSections options
);
```

Parameters

messageId
Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

messagePart
An integer value that specifies the message part that should be retrieved. A value of zero specifies that the complete message should be returned. If the message is a multipart MIME message, message parts start with a value of one.

offset
The byte offset into the message. This parameter can be used in conjunction with the ***length*** argument to return a specific part of a message. If this argument is omitted or a value of zero is specified, the server will return data from the beginning of the message.

length
An integer passed by reference which should be initialized to the maximum number of bytes to be read, and will contain the size of the message when the function returns. To specify the entire message, from the offset specified by the ***offset*** parameter to the end of the message, initialize the ***length*** parameter to a value of -1.

options
An ImapSections enumeration value which specifies which section of the message to return.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **OpenMessage** method opens a message or a specific part of a multipart message in the current mailbox. The message data may also be limited a specific byte offset and length, which can be useful for previewing the contents. The client can then read the contents of the message using the **Read** method, and once all of the data has been read, the message should be closed by calling the **CloseMessage** method.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.OpenMessage Overload List

ImapClient.OpenMessage Method (Int32, Int32, Int32, ImapSections)

Open the specified message for reading.

[Visual Basic]

```
Overloads Public Function OpenMessage( _
   ByVal messageId As Integer, _
   ByVal messagePart As Integer, _
   ByRef length As Integer, _
   ByVal options As ImapSections _
) As Boolean
```

[C#]

```
public bool OpenMessage(
   int messageId,
   int messagePart,
   ref int length,
   ImapSections options
);
```

Parameters

messageId
: Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

messagePart
: An integer value that specifies the message part that should be retrieved. A value of zero specifies that the complete message should be returned. If the message is a multipart MIME message, message parts start with a value of one.

length
: An integer passed by reference which should be initialized to the maximum number of bytes to be read, and will contain the size of the message when the function returns. To specify the entire message, initialize the ***length*** parameter to a value of -1.

options
: An ImapSections enumeration value which specifies which section of the message to return.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **OpenMessage** method opens a message or a specific part of a multipart message in the current mailbox. The message data may also be limited a specific byte offset and length, which can be useful for previewing the contents. The client can then read the contents of the message using the **Read** method, and once all of the data has been read, the message should be closed by calling the **CloseMessage** method.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.OpenMessage Overload List

ImapClient.Read Method

Read data from the server and store it in a byte array.

Overload List

Read data from the server and store it in a byte array.

public int Read(byte[]);

Read data from the server and store it in a byte array.

public int Read(byte[],int);

Read data from the server and store it in a string.

public int Read(ref string);

Read data from the server and store it in a string.

public int Read(ref string,int);

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Read Method (Byte[])

Read data from the server and store it in a byte array.

[Visual Basic]
```
Overloads Public Function Read( _
   ByVal buffer As Byte() _
) As Integer
```

[C#]
```
public int Read(
   byte[] buffer
);
```

Parameters

buffer

A byte array that the data will be stored in.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the size of the byte array passed to the method. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Read Overload List

ImapClient.Read Method (Byte[], Int32)

Read data from the server and store it in a byte array.

[Visual Basic]

```
Overloads Public Function Read( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Read(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array that the data will be stored in.

length

An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the number of bytes specified. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Read Overload List

ImapClient.Read Method (String)

Read data from the server and store it in a string.

[Visual Basic]

```
Overloads Public Function Read( _
   ByRef buffer As String _
) As Integer
```

[C#]

```
public int Read(
   ref string buffer
);
```

Parameters

buffer

A string that will contain the data read from the client.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to a maximum of 4096 bytes. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Read Overload List

ImapClient.Read Method (String, Int32)

Read data from the server and store it in a string.

[Visual Basic]

```
Overloads Public Function Read( _
   ByRef buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Read(
   ref string buffer,
   int length
);
```

Parameters

buffer

A string that will contain the data read from the client.

length

An integer value which specifies the maximum number of bytes of data to read.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the number of bytes specified. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Read Overload List

ImapClient.Refresh Method

Updates the list of available mailboxes.

[Visual Basic]
```
Public Function Refresh() As Boolean
```

[C#]
```
public bool Refresh();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Refresh** method updates the list of mailboxes that may be selected by the client. The available mailboxes may be enumerated using the **Mailbox** property array, with the **Mailboxes** property returning the total number of mailboxes.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.RenameMailbox Method

Change the name of a mailbox.

[Visual Basic]
```
Public Function RenameMailbox( _
   ByVal oldMailbox As String, _
   ByVal newMailbox As String _
) As Boolean
```

[C#]
```
public bool RenameMailbox(
   string oldMailbox,
   string newMailbox
);
```

Parameters

oldMailbox
A string that specifies the name of the mailbox to be renamed on the server. The mailbox must exist on the server, otherwise an error will be returned.

newMailbox
A string that specifies the new name for the mailbox. An error will be returned if a mailbox with that name already exists.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

If the existing mailbox name contains inferior hierarchical names (mailboxes under the specified mailbox) then those mailboxes will also be renamed. For example, if the mailbox "Mail/Pictures" contains two mailboxes, "Personal" and "Work" and it is renamed to "Mail/Images" then the two mailboxes under it would be automatically renamed to "Mail/Images/Personal" and "Mail/Images/Work".

If the mailbox being renamed is the currently selected mailbox, the current mailbox will be unselected and any messages marked for deletion will be expunged. The new mailbox name will then automatically be re-selected. To prevent deleted messages from being removed from the mailbox prior to being renamed, use the **UnselectMailbox** method to unselect the current mailbox before calling **RenameMailbox**. Note that if the rename operation fails, the client may be left in an unselected state.

It is permitted to rename the special mailbox INBOX. In this case, the messages will be moved from the INBOX mailbox to the new mailbox. If the INBOX mailbox is currently selected, the new mailbox will not automatically be selected. INBOX will remain the selected mailbox.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.ReselectMailbox Method

Reselects the current mailbox.

[Visual Basic]

```
Public Function ReselectMailbox() As Boolean
```

[C#]

```
public bool ReselectMailbox();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ReselectMailbox** method forces the current mailbox to be reselected and updates those properties which return information about the mailbox, such as the **MailboxFlags** property. Deleted messages are not expunged from the mailbox and remain marked for deletion.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. If a socket has been allocated, it will be released and any active connections will be terminated. All properties will be reset to their default values.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.SearchMailbox Method

Search the current mailbox for messages that match the specified criteria.

Overload List

Search the current mailbox for messages that match the specified criteria.

public int SearchMailbox(string,int[],int);

Search the current mailbox for messages that match the specified criteria and character set.

public int SearchMailbox(string,string,int[],int);

See Also

ImapClient Class | SocketTools Namespace

ImapClient.SearchMailbox Method (String, Int32[], Int32)

Search the current mailbox for messages that match the specified criteria.

[Visual Basic]

```
Overloads Public Function SearchMailbox( _
   ByVal criteria As String, _
   ByVal messageList As Integer(), _
   ByVal maxMessages As Integer _
) As Integer
```

[C#]

```
public int SearchMailbox(
   string criteria,
   int[] messageList,
   int maxMessages
);
```

Parameters

criteria

A string which consists of one or more keywords which are used to define the search criteria. The following keywords are recognized:

Keyword	Description
ANSWERED	Match those messages which have the **ImapFlags.flagAnswered** flag set.
BCC *address*	Match those messages which contain the specified address in the BCC header field.
BEFORE *date*	Match those messages which were added to the mailbox prior to the specified date.
BODY *string*	Match those messages where the body contains the specified string.
CC *address*	Match those messages which contain the specified address in the CC header field.
DELETED	Match those messages which have the **ImapFlags.flagDeleted** flag set.
DRAFT	Match those messages which have the **ImapFlags.flagDraft** flag set.
FLAGGED	Match those messages which have the **ImapFlags.flagUrgent** flag set.
FROM *address*	Match those messages which contain the specified address in the FROM header field.
HEADER *field string*	Match those messages which contain the string in the specified header field. If no string is specified, then all messages which contain the header will be matched.
LARGER *size*	Match those messages which are larger than the

	specified size in bytes.
NEW	Match those messages which have the **ImapFlags.flagRecent** flag set, but not the **ImapFlags.flagSeen** flag.
OLD	Match those messages which do not have the **ImapFlags.flagRecent** flag set.
ON *date*	Match those messages which were added on the specified date.
RECENT	Match those messages which have the **ImapFlags.flagRecent** flag set.
SEEN	Match those messages which have the **ImapFlags.flagSeen** flag set.
SENTBEFORE *date*	Match those messages whose Date header value is earlier than the specified date.
SENTON *date*	Match those messages whose Date header value is the same as the specified date.
SENTSINCE *date*	Match those messages whose Date header value is later than the specified date.
SINCE *date*	Match those messages added to the mailbox after the specified date.
SMALLER *size*	Match those messages which are smaller than the specified size in bytes.
SUBJECT *string*	Match those messages whose Subject header contains the specified string.
TEXT *string*	Match those messages whose headers or body contains the specified string.
TO *address*	Match those messages which contain the specified address in the TO header field.
UID *sequence*	Match those messages with unique identifiers in the sequence set.
UNANSWERED	Match those messages which do not have the **ImapFlags.flagAnswered** flag set.
UNDELETED	Match those messages which do not have the **ImapFlags.flagDeleted** flag set.
UNDRAFT	Match those messages which do not have the **ImapFlags.flagDraft** flag set.
UNFLAGGED	Match those messages which do not have the **ImapFlags.flagUrgent** flag set.
UNSEEN	Match those messages which do not have the **ImapFlags.flagUnseen** flag set.

messageList

An array of integers which will contain the message numbers of those messages which match the search criteria.

maxMessages

An integer value which specifies the maximum number of message numbers which can be returned in the ***messageList*** array. This value cannot be larger than the size of the array.

Return Value

The number of messages which were found to match the search criteria. If no messages match the criteria, then the return value will be zero. A return value of -1 indicates an error, and the specific error code can be determined by checking the value of the **LastError** property.

Remarks

The **SearchMailbox** method is used to search a mailbox for messages which match a given criteria and return a list of the matching message numbers. The search criteria is composed of one or more search keywords and and optional value to match against. String searches are not case sensitive and partial matches in the message are returned. The message numbers returned by this method are only valid until the mailbox is expunged or another mailbox is selected.

In addition to the listed keywords, the keyword NOT may prefix a keyword to return those messages which do not match the search criteria. For example, "NOT TO user@domain.com" would return those messages which were not addressed to user@domain.com.

If multiple search keywords are specified, the result is the intersection of all those messages which meet the search criteria. For example, a search criteria of "DELETED SINCE 1-Jan-2010" would return all those messages which are marked for deletion and were added to the mailbox after 1 January 2010.

Those search keywords which expect dates must be specified in format ***dd-mmm-yyyy*** where the month is the three letter abbreviation for the month name. Note that the internal date the message was added to the mailbox is not the same as the value of the Date header field in the message.

If the search keyword requires a string value and the string contains one or more spaces, you must enclose the search string in quotes as part of the criteria string. The quotes around the search string prevents the server from interpreting it as a multiple search criteria to be evaluated. If you are using a search string provided by a user, it is recommended that you always enclose it in quotes to prevent any potential ambiguity in the search. Even if the search string does not contain any spaces, it is always safe to enclose it in quotes.

The UID keyword expects a one or more unique message identifiers. These values may provided as comma separated list, or a range delimited by a colon. For example, "UID 23000:24000" would return all those messages who have UIDs ranging from 23000 through to 24000.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.SearchMailbox Overload List

ImapClient.SearchMailbox Method (String, String, Int32[], Int32)

Search the current mailbox for messages that match the specified criteria and character set.

[Visual Basic]

```
Overloads Public Function SearchMailbox( _
   ByVal criteria As String, _
   ByVal charset As String, _
   ByVal messageList As Integer(), _
   ByVal maxMessages As Integer _
) As Integer
```

[C#]

```
public int SearchMailbox(
   string criteria,
   string charset,
   int[] messageList,
   int maxMessages
);
```

Parameters

criteria

A string which consists of one or more keywords which are used to define the search criteria. The following keywords are recognized:

Keyword	Description
ANSWERED	Match those messages which have the **ImapFlags.flagAnswered** flag set.
BCC *address*	Match those messages which contain the specified address in the BCC header field.
BEFORE *date*	Match those messages which were added to the mailbox prior to the specified date.
BODY *string*	Match those messages where the body contains the specified string.
CC *address*	Match those messages which contain the specified address in the CC header field.
DELETED	Match those messages which have the **ImapFlags.flagDeleted** flag set.
DRAFT	Match those messages which have the **ImapFlags.flagDraft** flag set.
FLAGGED	Match those messages which have the **ImapFlags.flagUrgent** flag set.
FROM *address*	Match those messages which contain the specified address in the FROM header field.
HEADER *field string*	Match those messages which contain the string in the specified header field. If no string is specified, then all messages which contain the header will be matched.

LARGER *size*	Match those messages which are larger than the specified size in bytes.
NEW	Match those messages which have the **ImapFlags.flagRecent** flag set, but not the **ImapFlags.flagSeen** flag.
OLD	Match those messages which do not have the **ImapFlags.flagRecent** flag set.
ON *date*	Match those messages which were added on the specified date.
RECENT	Match those messages which have the **ImapFlags.flagRecent** flag set.
SEEN	Match those messages which have the **ImapFlags.flagSeen** flag set.
SENTBEFORE *date*	Match those messages whose Date header value is earlier than the specified date.
SENTON *date*	Match those messages whose Date header value is the same as the specified date.
SENTSINCE *date*	Match those messages whose Date header value is later than the specified date.
SINCE *date*	Match those messages added to the mailbox after the specified date.
SMALLER *size*	Match those messages which are smaller than the specified size in bytes.
SUBJECT *string*	Match those messages whose Subject header contains the specified string.
TEXT *string*	Match those messages whose headers or body contains the specified string.
TO *address*	Match those messages which contain the specified address in the TO header field.
UID *sequence*	Match those messages with unique identifiers in the sequence set.
UNANSWERED	Match those messages which do not have the **ImapFlags.flagAnswered** flag set.
UNDELETED	Match those messages which do not have the **ImapFlags.flagDeleted** flag set.
UNDRAFT	Match those messages which do not have the **ImapFlags.flagDraft** flag set.
UNFLAGGED	Match those messages which do not have the **ImapFlags.flagUrgent** flag set.
UNSEEN	Match those messages which do not have the **ImapFlags.flagUnseen** flag set.

charset

An string which specifies the character set to use when searching the mailbox. If this argument is omitted, the default US-ASCII character set will be used. Note that not all servers support searches using anything but the default character set.

messageList

An array of integers which will contain the message numbers of those messages which match the search criteria.

maxMessages

An integer value which specifies the maximum number of message numbers which can be returned in the ***messageList*** array. This value cannot be larger than the size of the array.

Return Value

The number of messages which were found to match the search criteria. If no messages match the criteria, then the return value will be zero. A return value of -1 indicates an error, and the specific error code can be determined by checking the value of the **LastError** property.

Remarks

The **SearchMailbox** method is used to search a mailbox for messages which match a given criteria and return a list of the matching message numbers. The search criteria is composed of one or more search keywords and and optional value to match against. String searches are not case sensitive and partial matches in the message are returned. The message numbers returned by this method are only valid until the mailbox is expunged or another mailbox is selected.

In addition to the listed keywords, the keyword NOT may prefix a keyword to return those messages which do not match the search criteria. For example, "NOT TO user@domain.com" would return those messages which were not addressed to user@domain.com.

If multiple search keywords are specified, the result is the intersection of all those messages which meet the search criteria. For example, a search criteria of "DELETED SINCE 1-Jan-2010" would return all those messages which are marked for deletion and were added to the mailbox after 1 January 2010.

Those search keywords which expect dates must be specified in format ***dd-mmm-yyyy*** where the month is the three letter abbreviation for the month name. Note that the internal date the message was added to the mailbox is not the same as the value of the Date header field in the message.

If the search keyword requires a string value and the string contains one or more spaces, you must enclose the search string in quotes as part of the criteria string. The quotes around the search string prevents the server from interpreting it as a multiple search criteria to be evaluated. If you are using a search string provided by a user, it is recommended that you always enclose it in quotes to prevent any potential ambiguity in the search. Even if the search string does not contain any spaces, it is always safe to enclose it in quotes.

The UID keyword expects a one or more unique message identifiers. These values may provided as comma separated list, or a range delimited by a colon. For example, "UID 23000:24000" would return all those messages who have UIDs ranging from 23000 through to 24000.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.SearchMailbox Overload List

ImapClient.SelectMailbox Method

Selects the specified mailbox for read-write access.

```
[Visual Basic]
Public Function SelectMailbox( _
   ByVal mailboxName As String _
) As Boolean
```

```
[C#]
public bool SelectMailbox(
   string mailboxName
);
```

Parameters

mailboxName

A string argument which specifies the name of the mailbox to be selected.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SelectMailbox** method is used to select a mailbox in read-write mode. If the client has a different mailbox currently selected, that mailbox will be closed and any messages marked for deletion will be expunged. To prevent deleted messages from being removed from the previous mailbox, use the **UnselectMailbox** method prior to selecting the new mailbox.

The special case-insensitive mailbox name INBOX is used for new messages. Other mailbox names may or may not be case-sensitive depending on the IMAP server's operating system and implementation.

To access a mailbox in read-only mode, use the **ExamineMailbox** method.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.StoreMessage Method

Retrieve a message from the current mailbox and store it in a file on the local system.

Overload List

Retrieve a message from the current mailbox and store it in a file on the local system.

public bool StoreMessage(int,string);

Retrieve the current message and store it in a file on the local system.

public bool StoreMessage(string);

See Also

ImapClient Class | SocketTools Namespace

ImapClient.StoreMessage Method (Int32, String)

Retrieve a message from the current mailbox and store it in a file on the local system.

[Visual Basic]

```
Overloads Public Function StoreMessage( _
   ByVal messageId As Integer, _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool StoreMessage(
   int messageId,
   string fileName
);
```

Parameters

messageId

Number of message to retrieve. This value must be greater than zero. The first message in the mailbox is message number one.

fileName

A string which specifies the file that the message will be stored in. If the file does not exist, it will be created. If the file does exist, it will be overwritten with the contents of the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **StoreMessage** method retrieves a message from the server and stores it in a file on the local system. The contents of the message is stored as a text file, using the specified file name. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.StoreMessage Overload List

ImapClient.StoreMessage Method (String)

Retrieve the current message and store it in a file on the local system.

```
[Visual Basic]
Overloads Public Function StoreMessage( _
   ByVal fileName As String _
) As Boolean
```

```
[C#]
public bool StoreMessage(
   string fileName
);
```

Parameters

fileName

A string which specifies the file that the message will be stored in. If the file does not exist, it will be created. If the file does exist, it will be overwritten with the contents of the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **StoreMessage** method retrieves the current message from the server and stores it in a file on the local system. The contents of the message is stored as a text file, using the specified file name. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The current message number is specified by the value of the **Message** property.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.StoreMessage Overload List

ImapClient.SubscribeMailbox Method

Subscribes the user to the current mailbox.

Overload List

Subscribes the user to the current mailbox.

public bool SubscribeMailbox();

Subscribes the user to the specified mailbox.

public bool SubscribeMailbox(string);

See Also

ImapClient Class | SocketTools Namespace

ImapClient.SubscribeMailbox Method ()

Subscribes the user to the current mailbox.

[Visual Basic]

```
Overloads Public Function SubscribeMailbox() As Boolean
```

[C#]

```
public bool SubscribeMailbox();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SubscribeMailbox** method adds the current mailbox to the current user's list of active or subscribed mailboxes. The user will remain subscribed to the mailbox across multiple sessions, until the **UnsubscribeMailbox** method is called to remove the mailbox from the subscription list.

Note that if a user subscribes to a mailbox and that mailbox is later renamed or deleted, the mailbox will not be automatically removed from the user's subscription list. To determine if the current mailbox is in the user's subscription list, check the **Subscribed** property.

The current mailbox is specified by the value of the **MailboxName** property.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.SubscribeMailbox Overload List

ImapClient.SubscribeMailbox Method (String)

Subscribes the user to the specified mailbox.

[Visual Basic]

```
Overloads Public Function SubscribeMailbox( _
   ByVal mailboxName As String _
) As Boolean
```

[C#]

```
public bool SubscribeMailbox(
   string mailboxName
);
```

Parameters

mailboxName

A string which specifies the name of the mailbox to subscribe to.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SubscribeMailbox** method adds the specified mailbox to the current user's list of active or subscribed mailboxes. The user will remain subscribed to the mailbox across multiple sessions, until the **UnsubscribeMailbox** method is called to remove the mailbox from the subscription list.

Note that if a user subscribes to a mailbox and that mailbox is later renamed or deleted, the mailbox will not be automatically removed from the user's subscription list. To determine if the current mailbox is in the user's subscription list, check the **Subscribed** property.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.SubscribeMailbox Overload List

ImapClient.UndeleteMessage Method

Removes the deletion flag for the specified message.

[Visual Basic]

```
Public Function UndeleteMessage( _
   ByVal messageId As Integer _
) As Boolean
```

[C#]

```
public bool UndeleteMessage(
   int messageId
);
```

Parameters

messageId

Number of message to undelete from the server. This value must be greater than zero. The first message in the mailbox is message number one.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **UndeleteMessage** method removes the deletion flag for the specified message in the current mailbox. To determine if a message has been marked for deletion, set the **Message** property to the message number and then check the value of the **MessageFlags** property to determine if the **imapFlagDeleted** bit flag has been set.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Uninitialize Method

Uninitialize the class library and release any resources allocated for the current thread.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method terminates any active connection, releases resources allocated for the current thread and unloads the networking library. After this method has been called, no further client operations may be performed until the class instance has been re-initialized.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the class is no longer needed.

See Also

ImapClient Class | SocketTools Namespace | Initialize Method

ImapClient.UnselectMailbox Method

Unselects the current mailbox and expunges deleted messages.

Overload List

Unselects the current mailbox and expunges deleted messages.

public bool UnselectMailbox();

Unselects the current mailbox.

public bool UnselectMailbox(bool);

See Also

ImapClient Class | SocketTools Namespace

ImapClient.UnselectMailbox Method ()

Unselects the current mailbox and expunges deleted messages.

[Visual Basic]

```
Overloads Public Function UnselectMailbox() As Boolean
```

[C#]

```
public bool UnselectMailbox();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **UnselectMailbox** method unselects the current mailbox. Once the mailbox has been unselected, no messages in that mailbox can be accessed, and any messages which have been flagged for deletion are removed.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.UnselectMailbox Overload List

ImapClient.UnselectMailbox Method (Boolean)

Unselects the current mailbox.

[Visual Basic]
```
Overloads Public Function UnselectMailbox( _
   ByVal expunge As Boolean _
) As Boolean
```

[C#]
```
public bool UnselectMailbox(
   bool expunge
);
```

Parameters

expunge

An boolean value which determines if deleted messages will be expunged from the mailbox. A value of **true** specifies that messages that have been marked for deletion will be removed from the mailbox. A value of **false** specifies that no messages will be removed from the mailbox.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **UnselectMailbox** method unselects the current mailbox. Once the mailbox has been unselected, no messages in that mailbox can be accessed.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.UnselectMailbox Overload List

ImapClient.UnsubscribeMailbox Method

Unsubscribes the user from the current mailbox.

Overload List

Unsubscribes the user from the current mailbox.

public bool UnsubscribeMailbox();

Unsubscribes the user from the specified mailbox.

public bool UnsubscribeMailbox(string);

See Also

ImapClient Class | SocketTools Namespace

ImapClient.UnsubscribeMailbox Method ()

Unsubscribes the user from the current mailbox.

[Visual Basic]

```
Overloads Public Function UnsubscribeMailbox() As Boolean
```

[C#]

```
public bool UnsubscribeMailbox();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **UnsubscribeMailbox** method removes the current mailbox from the current user's list of active or subscribed mailboxes. To determine if the current mailbox is in the user's subscription list, check the **Subscribed** property.

The current mailbox is specified by the value of the **MailboxName** property.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.UnsubscribeMailbox Overload List

ImapClient.UnsubscribeMailbox Method (String)

Unsubscribes the user from the specified mailbox.

[Visual Basic]

```
Overloads Public Function UnsubscribeMailbox( _
   ByVal mailboxName As String _
) As Boolean
```

[C#]

```
public bool UnsubscribeMailbox(
   string mailboxName
);
```

Parameters

mailboxName

A string which specifies the name of the mailbox to unsubscribe from.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **UnsubscribeMailbox** method removes the specified mailbox from the current user's list of active or subscribed mailboxes. To determine if the current mailbox is in the user's subscription list, check the **Subscribed** property.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.UnsubscribeMailbox Overload List

ImapClient.Write Method

Write one or more bytes of data to the server.

Overload List

Write one or more bytes of data to the server.

public int Write(byte[]);

Write one or more bytes of data to the server.

public int Write(byte[],int);

Write a string of characters to the server.

public int Write(string);

Write a string of characters to the server.

public int Write(string,int);

See Also

ImapClient Class | SocketTools Namespace

ImapClient.Write Method (Byte[])

Write one or more bytes of data to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Write(
   byte[] buffer
);
```

Parameters

buffer

A byte array that contains the data to be written to the server.

Return Value

An integer value which specifies the number of bytes actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Write Overload List

ImapClient.Write Method (Byte[], Int32)

Write one or more bytes of data to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Write(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array that contains the data to be written to the server.

length

An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Write Overload List

ImapClient.Write Method (String)

Write a string of characters to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String _
) As Integer
```

[C#]

```
public int Write(
   string buffer
);
```

Parameters

buffer
A string which contains the data to be written to the server.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Write Overload List

ImapClient.Write Method (String, Int32)

Write a string of characters to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Write(
   string buffer,
   int length
);
```

Parameters

buffer

A string which contains the data to be written to the server.

length

An integer value which specifies the maximum number of characters to write. This value cannot be larger than the length of the string specified by the caller.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

ImapClient Class | SocketTools Namespace | ImapClient.Write Overload List

ImapClient Events

The events of the **ImapClient** class are listed below. For a complete list of **ImapClient** class members, see the ImapClient Members topic.

Public Instance Events

⚡ OnCancel	Occurs when a blocking client operation is canceled.
⚡ OnConnect	Occurs when a connection is established with the remote host.
⚡ OnDisconnect	Occurs when the remote host disconnects from the local system.
⚡ OnError	Occurs when an client operation fails.
⚡ OnProgress	Occurs as a data stream is being read or written to the client.
⚡ OnRead	Occurs when data is available to be read from the client.
⚡ OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
⚡ OnUpdate	Occurs when the server sends a mailbox update notification to the client.
⚡ OnWrite	Occurs when data can be written to the client.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.OnCancel Event

Occurs when a blocking client operation is canceled.

[Visual Basic]

```
Public Event OnCancel As EventHandler
```

[C#]

```
public event EventHandler OnCancel;
```

Remarks

The **OnCancel** event is generated when a blocking client operation, such as sending or receiving data, is canceled with the **Cancel** method. To assist in determining which operation was canceled, check the value of the **Status** property.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.OnConnect Event

Occurs when a connection is established with the remote host.

[Visual Basic]

```
Public Event OnConnect As EventHandler
```

[C#]

```
public event EventHandler OnConnect;
```

Remarks

The **OnConnect** event occurs when a connection is made with a remote host as a result of a **Connect** method call. When the **Connect** method is called and the **Blocking** property is set to **false**, a socket is created but the connection is not actually established until after this event occurs. Between the time connection process is started and this event fires, no operation may be performed on the client other than calling the **Disconnect** method.

This event is only generated if the client is in non-blocking mode.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.OnDisconnect Event

Occurs when the remote host disconnects from the local system.

[Visual Basic]

```
Public Event OnDisconnect As EventHandler
```

[C#]

```
public event EventHandler OnDisconnect;
```

Remarks

The **OnDisconnect** event occurs when the remote host closes its connection, terminating the client session with the application. Because there may still be data in the client receive buffers, you should continue to read data from the client until the **Read** method returns a value of 0. Once all of the data has been read, you should call the **Disconnect** method to close the local socket and release the resources allocated for the client.

This event is only generated if the client is in non-blocking mode.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.OnError Event

Occurs when an client operation fails.

[Visual Basic]
```
Public Event OnError As OnErrorEventHandler
```

[C#]
```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type ImapClient.ErrorEventArgs containing data related to this event. The following **ImapClient.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a client operation fails.

See Also

ImapClient Class | SocketTools Namespace | OnErrorEventHandler Delegate

ImapClient.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see ImapClient.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.ImapClient.ErrorEventArgs

[Visual Basic]
```
Public Class ImapClient.ErrorEventArgs
    Inherits EventArgs
```

[C#]
```
public class ImapClient.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

ImapClient.ErrorEventArgs Members | SocketTools Namespace

ImapClient.ErrorEventArgs Members

ImapClient.ErrorEventArgs overview

Public Instance Constructors

ImapClient.ErrorEventArgs Constructor	Initializes a new instance of the ImapClient.ErrorEventArgs class.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

ImapClient.ErrorEventArgs Class | SocketTools Namespace

ImapClient.ErrorEventArgs Constructor

Initializes a new instance of the ImapClient.ErrorEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public ImapClient.ErrorEventArgs();
```

See Also

ImapClient.ErrorEventArgs Class | SocketTools Namespace

ImapClient.ErrorEventArgs Properties

The properties of the **ImapClient.ErrorEventArgs** class are listed below. For a complete list of **ImapClient.ErrorEventArgs** class members, see the ImapClient.ErrorEventArgs Members topic.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

ImapClient.ErrorEventArgs Class | SocketTools Namespace

ImapClient.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

ImapClient.ErrorEventArgs Class | SocketTools Namespace | Error Property

ImapClient.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public ImapClient.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

ImapClient.ErrorEventArgs Class | SocketTools Namespace | Description Property

ImapClient.OnProgress Event

Occurs as a data stream is being read or written to the client.

[Visual Basic]

```
Public Event OnProgress As OnProgressEventHandler
```

[C#]

```
public event OnProgressEventHandler OnProgress;
```

Event Data

The event handler receives an argument of type ImapClient.ProgressEventArgs containing data related to this event. The following **ImapClient.ProgressEventArgs** properties provide information specific to this event.

Property	Description
BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
BytesTotal	Gets a value which specifies the total number of bytes in the data stream.
Message	Gets the message number.
Percent	Gets a value which specifies the percentage of data that has been read or written.

Remarks

The **OnProgress** event occurs as a data stream is being read or written to the client. If large amounts of data are being read or written, this event can be used to update a progress bar or other user-interface component to provide the user with some visual feedback on the progress of the operation.

See Also

ImapClient Class | SocketTools Namespace | OnProgressEventHandler Delegate

ImapClient.ProgressEventArgs Class

Provides data for the OnProgress event.

For a list of all members of this type, see ImapClient.ProgressEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.ImapClient.ProgressEventArgs

[Visual Basic]

```
Public Class ImapClient.ProgressEventArgs
    Inherits EventArgs
```

[C#]

```
public class ImapClient.ProgressEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ProgressEventArgs specifies the number of bytes copied from the data stream, the total number of bytes in the data stream and a completion percentage.

The OnProgress event occurs as a data stream is being read or written to the client. This event only occurs when the **GetHeaders**, **GetMessage** or **StoreMessage** methods are called.

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

ImapClient.ProgressEventArgs Members | SocketTools Namespace

ImapClient.ProgressEventArgs Members

ImapClient.ProgressEventArgs overview

Public Instance Constructors

ImapClient.ProgressEventArgs Constructor	Initializes a new instance of the ImapClient.ProgressEventArgs class.

Public Instance Properties

BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
BytesTotal	Gets a value which specifies the total number of bytes in the data stream.
Message	Gets the message number.
Percent	Gets a value which specifies the percentage of data that has been read or written.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

ImapClient.ProgressEventArgs Class | SocketTools Namespace

ImapClient.ProgressEventArgs Constructor

Initializes a new instance of the ImapClient.ProgressEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public ImapClient.ProgressEventArgs();
```

See Also

ImapClient.ProgressEventArgs Class | SocketTools Namespace

ImapClient.ProgressEventArgs Properties

The properties of the **ImapClient.ProgressEventArgs** class are listed below. For a complete list of **ImapClient.ProgressEventArgs** class members, see the ImapClient.ProgressEventArgs Members topic.

Public Instance Properties

BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
BytesTotal	Gets a value which specifies the total number of bytes in the data stream.
Message	Gets the message number.
Percent	Gets a value which specifies the percentage of data that has been read or written.

See Also

ImapClient.ProgressEventArgs Class | SocketTools Namespace

ImapClient.ProgressEventArgs.BytesCopied Property

Gets a value which specifies the number of bytes of data that has been read or written.

[Visual Basic]

```
Public ReadOnly Property BytesCopied As Integer
```

[C#]

```
public int BytesCopied {get;}
```

Property Value

An integer value which specifies the number of bytes of data.

Remarks

The **BytesCopied** property specifies the number of bytes that have been read from the client and stored in the local stream buffer, or written from the stream buffer to the client.

See Also

ImapClient.ProgressEventArgs Class | SocketTools Namespace | BytesTotal Property | Percent Property

ImapClient.ProgressEventArgs.BytesTotal Property

Gets a value which specifies the total number of bytes in the data stream.

[Visual Basic]

```
Public ReadOnly Property BytesTotal As Integer
```

[C#]

```
public int BytesTotal {get;}
```

Property Value

An integer value which specifies the number of bytes of data.

Remarks

The **BytesTotal** property specifies the total amount of data being read from the client and stored in the data stream, or written from the data stream to the client. If the amount of data was unknown or unspecified at the time the method call was made, then this value will always be the same as the **BytesCopied** property.

See Also

ImapClient.ProgressEventArgs Class | SocketTools Namespace | BytesCopied Property | Percent Property

ImapClient.ProgressEventArgs.Message Property

Gets the message number.

[Visual Basic]

```
Public ReadOnly Property Message As Integer
```

[C#]

```
public int Message {get;}
```

Property Value

An integer value which specifies the message number.

Remarks

The **Message** property specifies the message number for the current message that is being downloaded from the mail server to the local host. If the **OnProgress** event occurs while message data is being uploaded to the server, this property will return a value of zero.

See Also

ImapClient.ProgressEventArgs Class | SocketTools Namespace

ImapClient.ProgressEventArgs.Percent Property

Gets a value which specifies the percentage of data that has been read or written.

[Visual Basic]

```
Public ReadOnly Property Percent As Integer
```

[C#]

```
public int Percent {get;}
```

Property Value

An integer value which specifies a percentage.

Remarks

The **Percent** property specifies the percentage of data that has been transmitted, expressed as an integer value between 0 and 100, inclusive. If the maximum size of the data stream was not specified by the caller, this value will always be 100.

See Also

ImapClient.ProgressEventArgs Class | SocketTools Namespace | BytesCopied Property | BytesTotal Property

ImapClient.OnRead Event

Occurs when data is available to be read from the client.

[Visual Basic]

```
Public Event OnRead As EventHandler
```

[C#]

```
public event EventHandler OnRead;
```

Remarks

The **OnRead** event occurs when data is available to be read from the client. This event is level-triggered, which means that once this event fires, it will not occur again until some data has been read from the client. This design prevents an application from being flooded with event notifications. It is recommended that your application read all of the available data from the server and store it in a local buffer for processing. See the example below.

This event is only generated if the client is in non-blocking mode.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.OnTimeout Event

Occurs when a blocking operation fails to complete before the timeout period elapses.

[Visual Basic]
```
Public Event OnTimeout As EventHandler
```

[C#]
```
public event EventHandler OnTimeout;
```

Remarks

The **OnTimeout** event occurs when a blocking operation, such as sending or receiving data on the client, fails to complete before the specified timeout period elapses. The timeout period for a blocking operation can be adjusted by setting the **Timeout** property.

This event is only generated if the client is in blocking mode.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.OnUpdate Event

Occurs when the server sends a mailbox update notification to the client.

[Visual Basic]

```
Public Event OnUpdate As OnUpdateEventHandler
```

[C#]

```
public event OnUpdateEventHandler OnUpdate;
```

Event Data

The event handler receives an argument of type ImapClient.UpdateEventArgs containing data related to this event. The following **ImapClient.UpdateEventArgs** properties provide information specific to this event.

Property	Description
Message	Gets the message number.
UpdateType	Gets the type of update notification that has been sent by the server.

Remarks

This event is only generated when the **Idle** method has been used to enable mailbox status monitoring.

See Also

ImapClient Class | SocketTools Namespace | Idle Method | OnUpdateEventHandler Delegate

ImapClient.UpdateEventArgs Class

Provides data for the OnUpdate event.

For a list of all members of this type, see ImapClient.UpdateEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.ImapClient.UpdateEventArgs

[Visual Basic]
```
Public Class ImapClient.UpdateEventArgs
    Inherits EventArgs
```

[C#]
```
public class ImapClient.UpdateEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

ImapClient.UpdateEventArgs Members | SocketTools Namespace

ImapClient.UpdateEventArgs Members

ImapClient.UpdateEventArgs overview

Public Instance Constructors

ImapClient.UpdateEventArgs Constructor	Initializes a new instance of the ImapClient.UpdateEventArgs class.

Public Instance Properties

Message	Gets the message number.
UpdateType	Gets the type of update notification that has been sent by the server.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

ImapClient.UpdateEventArgs Class | SocketTools Namespace

ImapClient.UpdateEventArgs Constructor

Initializes a new instance of the ImapClient.UpdateEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public ImapClient.UpdateEventArgs();
```

See Also

ImapClient.UpdateEventArgs Class | SocketTools Namespace

ImapClient.UpdateEventArgs Properties

The properties of the **ImapClient.UpdateEventArgs** class are listed below. For a complete list of **ImapClient.UpdateEventArgs** class members, see the ImapClient.UpdateEventArgs Members topic.

Public Instance Properties

Message	Gets the message number.
UpdateType	Gets the type of update notification that has been sent by the server.

See Also

ImapClient.UpdateEventArgs Class | SocketTools Namespace

ImapClient.UpdateEventArgs.Message Property

Gets the message number.

[Visual Basic]
```
Public ReadOnly Property Message As Integer
```

[C#]
```
public int Message {get;}
```

Property Value

An integer value which specifies the message number.

Remarks

The **Message** property specifies the message number for the message that has been added to or expunged from the current mailbox. A value of zero indicates that the update notification does not reference a specific message in the mailbox.

See Also

ImapClient.UpdateEventArgs Class | SocketTools Namespace

ImapClient.UpdateEventArgs.UpdateType Property

Gets the type of update notification that has been sent by the server.

[Visual Basic]

```
Public ReadOnly Property UpdateType As IdleUpdate
```

[C#]

```
public ImapClient.IdleUpdate UpdateType {get;}
```

Property Value

One or more of the IdleUpdate enumeration flags.

See Also

ImapClient.UpdateEventArgs Class | SocketTools Namespace

ImapClient.OnWrite Event

Occurs when data can be written to the client.

[Visual Basic]
```
Public Event OnWrite As EventHandler
```

[C#]
```
public event EventHandler OnWrite;
```

Remarks

The **OnWrite** event occurs when the application can write data to the client. This event will typically occur when a connection is first established with the remote host, and after the **Write** method has failed because there was insufficient memory available in the client send buffers. In the second case, when some of the buffered data has been successfully sent to the remote host and there is space available in the send buffers, this event is used to signal the application that it may attempt to send more data.

This event is only generated if the client is in non-blocking mode.

See Also

ImapClient Class | SocketTools Namespace

ImapClient.ErrorCode Enumeration

Specifies the error codes returned by the ImapClient class.

[Visual Basic]

```
Public Enum ImapClient.ErrorCode
```

[C#]

```
public enum ImapClient.ErrorCode
```

Remarks

The ImapClient class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.
errorProxyAuthenticationRequired	Proxy authentication required.

errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.
errorUnsupportedMediaType	Request specified an unsupported media type.

errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.
errorNoMoreMailboxes	No more mailboxes exist on this server.

errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.
errorInvalidThreadHandle	The specified thread handle is no longer valid.

errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been defined.

errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or unrecognized.

errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

ImapClient.ImapFlags Enumeration

Specifies the mailbox and message flags that the ImapClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum ImapClient.ImapFlags
```

[C#]
```
[Flags]
public enum ImapClient.ImapFlags
```

Members

Member Name	Description	Value
flagNone	No flags have been set for the current message or mailbox.	0
flagAnswered	The message has been answered.	1
flagDraft	The message is a draft copy and has not been delivered.	2
flagUrgent	The message has been flagged for urgent or special attention.	4
flagSeen	The message has been read.	8
flagRecent	The message has been added to the mailbox recent.	256
flagDeleted	The message has been marked for deletion.	512
flagNoInferiors	The mailbox does not contain any child mailboxes. In the IMAP protocol, these are referred to as inferior hierarchical mailbox names.	65536
flagNoSelect	The mailbox cannot be selected or examined. This flag is typically used by servers to indicate that the mailbox name refers to a directory on the server, not an actual mailbox.	131072
flagMarked	The mailbox is marked as being of interest to a client. If this flag is used, it typically means that the mailbox contains messages. An application should not depend on this flag being present for any given mailbox. Some IMAP servers do not support marked or unmarked flags for mailboxes.	262144
flagUnmarked	The mailbox is marked as not being of interest to a client. If this flag is used, it	524288

	typically means that the mailbox does not contain any messages. An application should not depend on this flag being present for any given mailbox. Some IMAP servers do not support marked or unmarked flags for mailboxes.	

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

SocketTools Namespace

ImapClient.IdleOptions Enumeration

Specifies the idle monitoring options that the ImapClient class supports.

[Visual Basic]

```
Public Enum ImapClient.IdleOptions
```

[C#]

```
public enum ImapClient.IdleOptions
```

Remarks

The **Idle** method can operate in two modes, based on the options specified by the caller. If the option **imapIdleNoWait** is specified, the method begins monitoring the client session asynchronously and returns control immediately to the caller. If the server sends a update notification to the client, the **OnUpdate** event will fire with information about the status change. If the option **imapIdleWait** is specified, the method will block waiting for the server to send a notification message to the client. The method will return when either a message is received or the timeout period is exceeded.

Members

Member Name	Description
idleNoWait	The **Idle** method should return immediately after idle processing has been enabled. When this option is used, the application may continue to perform other functions while the client session is monitored for status updates sent by the server. The client will continue to monitor status changes until an IMAP command issued or the client disconnects from the server. This is the default option.
idleWait	The **Idle** method should wait until the server sends a status update, or until the timeout period is reached. The client will stop monitoring status changes when the function returns. If this option is used in a single-threaded application, normal message processing can be impeded, causing the application to appear non-responsive until the timeout period is reached. It is strongly recommended that single-threaded applications with a user interface specify the **imapIdleNoWait** option instead.

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

SocketTools Namespace | Idle Method (SocketTools.ImapClient) | OnUpdate Event (SocketTools.ImapClient)

ImapClient.IdleUpdate Enumeration

Specifies the types of update notification messages that the ImapClient class supports.

[Visual Basic]
```
Public Enum ImapClient.IdleUpdate
```

[C#]
```
public enum ImapClient.IdleUpdate
```

Members

Member Name	Description
updateUnknown	The server has sent an unrecognized notification message. This does not necessarily reflect an error condition, as some servers may send additional notification messages beyond the standard EXISTS, EXPUNGE and RECENT messages. Most applications should ignore this type of notification.
updateMessage	The server has sent notification message to the client indicating that a new message has arrived. Typically this update notification occurs shortly after the new message has been stored in the current mailbox.
updateExpunge	The server has sent a notification message to the client indicating that a message has been removed from the current mailbox. It is recommended that the application re-examine the mailbox when this notification is received. Typically this notification is only sent periodically by the server, and may not be sent immediately after a message has been expunged from the mailbox.
updateMailbox	The server has sent notification message to the client indicating that the state of the mailbox has changed. This message is sent periodically by the server and may not be sent immediately after a new message arrives or a message is flagged as unread. It is recommended that the application re-examine the mailbox when this notification is received.

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

SocketTools Namespace | OnUpdate

ImapClient.ImapOptions Enumeration

Specifies the options that the ImapClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum ImapClient.ImapOptions
```

[C#]
```
[Flags]
public enum ImapClient.ImapOptions
```

Remarks

The ImapClient class uses the **ImapOptions** enumeration to specify one or more options to be used when establishing a connection with a remote host. Multiple options may be specified if necessary.

Members

Member Name	Description	Value
optionNone	No option specified.	0
optionDefault	The default connection option. This is the same as specifying **optionNone**.	0
optionIdentify	This option specifies the client should identify itself to the server. If enabled, the client will send the ID command to the server as defined in RFC 2971. This option has no effect if the server does not support the ID command.	1
optionTunnel	This option specifies that a tunneled TCP connection and/or port-forwarding is being used to establish the connection to the server. This changes the behavior of the client with regards to internal checks of the destination IP address and remote port number, default capability selection and how the connection is established. This option also forces all connections to be outbound and enables the firewall compatibility features in the client.	1024
optionTrustedSite	This option specifies the server is trusted. The server certificate will not be validated and the connection will always be permitted. This option only affects connections using either the SSL or TLS protocols.	2048
optionSecure	This option specifies the client should attempt to establish a secure	4096

	connection with the server. The server must support secure connections using either the SSL or TLS protocol.	
optionExplicitSSL	This option specifies the client should attempt to establish a secure explicit SSL session. The initial connection to the server is not encrypted, and the client will attempt to negotiate a secure connection by sending the STARTTLS command to the server. Some servers may require this option when connecting to the server on ports other than the default secure port of 993.	4096
optionImplicitSSL	This option specifies the client should attempt to establish a secure implicit SSL session. The SSL handshake is initiated immediately after the connection to the server has been established.	8192
optionSecureFallback	This option specifies the client should permit the use of less secure cipher suites for compatibility with legacy servers. If this option is specified, the client will permit connections using TLS 1.0 and cipher suites that use RC4, MD5 and SHA1.	32768
optionPreferIPv6	This option specifies the client should prefer the use of IPv6 if the server hostname can be resolved to both an IPv6 and IPv4 address. This option is ignored if the local system does not have IPv6 enabled, or when the hostname can only be resolved to an IPv4 address. If the server hostname can only be resolved to an IPv6 address, the client will attempt to establish a connection using IPv6 regardless if this option has been specified.	262144
optionFreeThread	This option specifies that class methods may be called from any thread, and not only the thread that established the connection. Using this option disables certain internal safety checks that are made by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads.	524288

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

SocketTools Namespace

ImapClient.ImapResult Enumeration

Specifies the result codes that the ImapClient class supports.

[Visual Basic]

```
Public Enum ImapClient.ImapResult
```

[C#]

```
public enum ImapClient.ImapResult
```

Members

Member Name	Description
resultUnknown	An unknown result code was returned by the server.
resultOk	The previous command completed successfully. The result string contains information about the results of the command.
resultNo	The previous command could not be completed. The result string contains information about why the command failed.
resultBad	The previous command could not be completed, the command may be invalid or not supported on the server. The result string contains information about why the command failed.
resultContinue	The command has executed and is waiting for additional data from the client.

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

SocketTools Namespace

ImapClient.ImapSections Enumeration

Specifies the message sections that the ImapClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum ImapClient.ImapSections
```

[C#]

```
[Flags]
public enum ImapClient.ImapSections
```

Members

Member Name	Description	Value
sectionDefault	All headers and the complete body of the specified message or message part are to be retrieved. The client application is responsible for parsing the header block. If the message is a MIME multipart message and the complete message is returned, the application is responsible for parsing the individual message parts if necessary.	0
sectionHeader	All headers for the specified message or message part are to be retrieved. The client application is responsible for parsing the header block.	1
sectionMimeHeader	The MIME headers for the specified message or message are to be retrieved. Only those header fields which are used in MIME messages, such as Content-Type will be returned to the client. This is typically useful when processing the header for a multipart message which contains file attachments. The client application is responsible for parsing the header block.	2
sectionBody	The body of the specified message or message part will be retrieved. For a MIME formatted message, this may include data that is uuencoded or base64 encoded. The application is responsible for decoding this data.	4
sectionPreview	The message header or body is being previewed and should not be marked as read by the server. This prevents the message from having the	4096

	IMAP_FLAG_SEEN flag from being automatically set when the message data is retrieved.	

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

SocketTools Namespace

ImapClient.ImapStatus Enumeration

Specifies the status values that may be returned by the ImapClient class.

[Visual Basic]
```
Public Enum ImapClient.ImapStatus
```

[C#]
```
public enum ImapClient.ImapStatus
```

Remarks

The ImapClient class uses the **ImapStatus** enumeration to identify the current status of the client.

Members

Member Name	Description
statusUnused	A client session has not been created. Attempts to perform any network operations, such as sending or receiving data, will generate an error.
statusIdle	A client session has been created, but is not currently in use. A blocking socket operation can be executed at this point.
statusConnect	The client is in the process of establishing a connection with a remote host.
statusRead	The client is in the process of receiving data from a remote host.
statusWrite	The client is in the process of sending data to a remote host.
statusDisconnect	The client session is being closed and subsequent attempts to access the client will result in an error.

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

SocketTools Namespace

ImapClient.SecureCipherAlgorithm Enumeration

Specifies the encryption algorithms that the ImapClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum ImapClient.SecureCipherAlgorithm
```

[C#]
```
[Flags]
public enum ImapClient.SecureCipherAlgorithm
```

Remarks

The ImapClient class uses the **SecureCipherAlgorithm** enumeration to identify which encryption algorithm was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
cipherNone	No cipher has been selected. A secure connection has not been established with the remote host.	0
cipherRC2	The RC2 block cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.	1
cipherRC4	The RC4 stream cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.	2
cipherRC5	The RC5 block cipher was selected. This is a variable key length cipher which supports keys up to 2040 bits, in 8-bit increments.	4
cipherDES	The DES (Data Encryption Standard) block cipher was selected. This is a fixed key length cipher using 56-bit keys.	8
cipherDES3	The Triple DES block cipher was selected. This cipher encrypts the data three times using different keys, effectively using a 168-bit key length.	16
cipherDESX	A variant of the DES block cipher which XORs an extra 64-bits of the key before and after the plaintext has been encrypted, increasing the key size to 184 bits.	32
cipherAES	The Advanced Encryption Standard	64

	cipher (also known as the Rijndael cipher) is a fixed block size cipher which use a key size of 128, 192 or 256 bits. This cipher is supported on Windows XP SP3 SP3 and later versions of the operating system.	
cipherSkipjack	The Skipjack block cipher was selected. This is a fixed key length cipher, using 80-bit keys.	128

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

SocketTools Namespace

ImapClient.SecureHashAlgorithm Enumeration

Specifies the hash algorithms that the ImapClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum ImapClient.SecureHashAlgorithm
```

[C#]

```
[Flags]
public enum ImapClient.SecureHashAlgorithm
```

Remarks

The ImapClient class uses the **SecureHashAlgorithm** enumeration to identify the message digest (hash) algorithm that was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
hashNone	No hash algorithm has been selected. This is not a secure connection with the server.	0
hashMD5	The MD5 algorithm was selected. This algorithm produces a 128-bit message digest. This algorithm is no longer considered to be cryptographically secure.	1
hashSHA	The SHA-1 algorithm was selected. This algorithm produces a 160-bit message digest. This algorithm is no longer considered to be cryptographically secure.	2
hashSHA256	The SHA-256 algorithm was selected. This algorithm produces a 256-bit message digest.	4
hashSHA384	The SHA-384 algorithm was selected. This algorithm produces a 384-bit message digest.	8
hashSHA512	The SHA-512 algorithm was selected. This algorithm produces a 512-bit message digest.	16

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

SocketTools Namespace

ImapClient.SecureKeyAlgorithm Enumeration

Specifies the key exchange algorithms that the ImapClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum ImapClient.SecureKeyAlgorithm
```

[C#]
```
[Flags]
public enum ImapClient.SecureKeyAlgorithm
```

Remarks

The ImapClient class uses the **SecureKeyAlgorithm** enumeration to identify the key exchange algorithm that was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
keyExchangeNone	No key exchange algorithm has been selected. This is not a secure connection with the server.	0
keyExchangeRSA	The RSA public key exchange algorithm has been selected.	1
keyExchangeKEA	The KEA public key exchange algorithm has been selected. This is an improved version of the Diffie-Hellman public key algorithm.	2
keyExchangeDH	The Diffie-Hellman public key exchange algorithm has been selected.	4
keyExchangeECDH	The Elliptic Curve Diffie-Hellman key exchange algorithm was selected. This is a variant of the Diffie-Hellman algorithm which uses elliptic curve cryptography. This key exchange algorithm is only supported on Windows XP SP3 SP3 and later versions of the operating system.	8

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

SocketTools Namespace

ImapClient.SecurityCertificate Enumeration

Specifies the security certificate status values that may be returned by the ImapClient class.

[Visual Basic]

```
Public Enum ImapClient.SecurityCertificate
```

[C#]

```
public enum ImapClient.SecurityCertificate
```

Remarks

The ImapClient class uses the **SecurityCertificate** enumeration to identify the current status of the certificate that was provided by the remote host when a secure connection was established.

Members

Member Name	Description
certificateNone	No certificate information is available. A secure connection was not established with the server.
certificateValid	The certificate is valid.
certificateNoMatch	The certificate is valid, however the domain name specified in the certificate does not match the domain name of the remote host. The application can examine the **CertificateSubject** property to determine the site the certificate was issued to.
certificateExpired	The certificate has expired and is no longer valid. The application can examine the **CertificateExpires** property to determine when the certificate expired.
certificateRevoked	The certificate has been revoked and is no longer valid. It is recommended that the application immediately terminate the connection if this status is returned.
certificateUntrusted	The certificate has not been issued by a trusted authority, or the certificate is not trusted on the local host. It is recommended that the application immediately terminate the connection if this status is returned.
certificateInvalid	The certificate is invalid. This typically indicates that the internal structure of the certificate is damaged. It is recommended that the application immediately terminate the connection if this status is returned.

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

SocketTools Namespace

ImapClient.SecurityProtocols Enumeration

Specifies the security protocols that the ImapClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum ImapClient.SecurityProtocols
```

[C#]
```
[Flags]
public enum ImapClient.SecurityProtocols
```

Remarks

The ImapClient class uses the **SecurityProtocols** enumeration to specify one or more security protocols to be used when establishing a connection with a remote host. Multiple protocols may be specified if necessary and the actual protocol used will be negotiated with the remote host. It is recommended that most applications use **protocolDefault** when creating a secure connection.

Members

Member Name	Description	Value
protocolNone	No security protocol will be used, a secure connection will not be established.	0
protocolSSL2	The SSL 2.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections.	1
protocolSSL3	The SSL 3.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections. In most cases, this protocol is only selected if TLS is not supported by the server.	2
protocolTLS10	The TLS 1.0 protocol has been selected. This version of the protocol is commonly used by older servers and is the only version of TLS supported on Windows platforms prior to Windows 7 SP1 and Windows Server 2008 R2.	4
protocolTLS11	The TLS 1.1 protocol has been selected. This version of TLS is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of the	8

	operating system.	
protocolTLS12	The TLS 1.2 protocol has been selected. This is the default version of the protocol and is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of Windows. It is recommended that you use this version of TLS.	16
protocolSSL	Any version of the Secure Sockets Layer (SSL) protocol should be used. The actual protocol version used will be negotiated with the remote host.	3
protocolTLS	Any version of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with with preference for TLS 1.2.	28
protocolTLS1	Version 1.0, 1.1 or 1.2 of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with preference for TLS 1.2.	28
protocolDefault	The default selection of security protocols will be used when establishing a connection. The TLS 1.2, 1.1 and 1.0 protocols will be negotiated with the host, in that order of preference. This option will always request the latest version of the preferred security protocols and is the recommended value.	16
protocolUnknown	An unknown or unsupported security protocol has been specified. This value indicates an error condition.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

SocketTools Namespace

ImapClient.TraceOptions Enumeration

Specifies the logging options that the ImapClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum ImapClient.TraceOptions
```

[C#]
```
[Flags]
public enum ImapClient.TraceOptions
```

Remarks

The ImapClient class uses the **TraceOptions** enumeration to specify what kind of debugging information is written to the trace logfile. These options are only meaningful when trace logging is enabled by setting the **Trace** property to **true**.

Members

Member Name	Description	Value
traceDefault	The default trace logging option. This is the same as specifying the **traceInfo** option.	0
traceInfo	All network function calls are written to the trace file. This is the default value.	0
traceError	Only those network function calls which fail are recorded in the trace file.	1
traceWarning	Only those network function calls which fail, or return values which indicate a warning, are recorded in the trace file.	2
traceHexDump	All network function calls are written to the trace file, plus all the data that is sent or received is displayed, in both ASCII and hexadecimal format.	4
traceProcess	All function calls in the current process are logged, rather than only those functions in the current thread. This option is useful for multithreaded applications that are using worker threads.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

SocketTools Namespace

ImapClient.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub ImapClient.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void ImapClient.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ErrorEventArgs that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

SocketTools Namespace

ImapClient.OnProgressEventHandler Delegate

Represents the method that will handle the OnProgress event.

[Visual Basic]

```
Public Delegate Sub ImapClient.OnProgressEventHandler( _
   ByVal sender As Object, _
   ByVal e As ProgressEventArgs _
)
```

[C#]

```
public delegate void ImapClient.OnProgressEventHandler(
      object sender,
      ProgressEventArgs e
   );
```

Parameters

sender
The source of the event.

e
A ProgressEventArgs that contains the event data.

Remarks

When you create an **OnProgressEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnProgressEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

SocketTools Namespace

ImapClient.OnUpdateEventHandler Delegate

Represents the method that will handle the OnUpdate event.

[Visual Basic]

```
Public Delegate Sub ImapClient.OnUpdateEventHandler( _
   ByVal sender As Object, _
   ByVal e As UpdateEventArgs _
)
```

[C#]

```
public delegate void ImapClient.OnUpdateEventHandler(
      object sender,
      UpdateEventArgs e
   );
```

Parameters

sender
The source of the event.

e
An UpdateEventArgs that contains the event data.

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

SocketTools Namespace | UpdateEventArgs Class

ImapClient.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see ImapClient.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.ImapClient.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class ImapClient.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class ImapClient.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the ImapClient class.

Example

```
<Assembly: SocketTools.ImapClient.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.ImapClient.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

ImapClient.RuntimeLicenseAttribute Members | SocketTools Namespace

ImapClient.RuntimeLicenseAttribute Members

ImapClient.RuntimeLicenseAttribute overview

Public Instance Constructors

ImapClient.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

ImapClient.RuntimeLicenseAttribute Class | SocketTools Namespace

ImapClient.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public ImapClient.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the ImapClient class.

See Also

ImapClient.RuntimeLicenseAttribute Class | SocketTools Namespace

ImapClient.RuntimeLicenseAttribute Properties

The properties of the **ImapClient.RuntimeLicenseAttribute** class are listed below. For a complete list of **ImapClient.RuntimeLicenseAttribute** class members, see the ImapClient.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

ImapClient.RuntimeLicenseAttribute Class | SocketTools Namespace

ImapClient.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

ImapClient.RuntimeLicenseAttribute Class | SocketTools Namespace

ImapClientException Class

The exception that is thrown when a client error occurs.

For a list of all members of this type, see ImapClientException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.ImapClientException

[Visual Basic]
```
Public Class ImapClientException
    Inherits ApplicationException
```

[C#]
```
public class ImapClientException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A ImapClientException is thrown by the ImapClient class when an error occurs.

The default constructor for the ImapClientException class sets the **ErrorCode** property to the last client error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.ImapClient (in SocketTools.ImapClient.dll)

See Also

ImapClientException Members | SocketTools Namespace

ImapClientException Members

ImapClientException overview

Public Instance Constructors

ImapClientException	Overloaded. Initializes a new instance of the ImapClientException class.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

ImapClientException Class | SocketTools Namespace

ImapClientException Constructor

Initializes a new instance of the ImapClientException class with the last network error code.

Overload List

Initializes a new instance of the ImapClientException class with the last network error code.

public ImapClientException();

Initializes a new instance of the ImapClientException class with a specified error number.

public ImapClientException(int);

Initializes a new instance of the ImapClientException class with a specified error message.

public ImapClientException(string);

Initializes a new instance of the ImapClientException class with a specified error message and a reference to the inner exception that is the cause of this exception.

public ImapClientException(string,Exception);

See Also

ImapClientException Class | SocketTools Namespace

ImapClientException Constructor ()

Initializes a new instance of the ImapClientException class with the last network error code.

[Visual Basic]

```
Overloads Public Sub New()
```

[C#]

```
public ImapClientException();
```

Remarks

The ctor constructor sets the **ErrorCode** property to the last client error that occurred. For more information about the errors that may occur, refer to the ImapClient.ErrorCode enumeration.

See Also

ImapClientException Class | SocketTools Namespace | ImapClientException Constructor Overload List

ImapClientException Constructor (String)

Initializes a new instance of the ImapClientException class with a specified error message.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]

```
public ImapClientException(
   string message
);
```

Parameters

message
: The error message that explains the reason for the exception.

Remarks

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

ImapClientException Class | SocketTools Namespace | ImapClientException Constructor Overload List

ImapClientException Constructor (String, Exception)

Initializes a new instance of the ImapClientException class with a specified error message and a reference to the inner exception that is the cause of this exception.

[Visual Basic]
```
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal innerException As Exception _
)
```

[C#]
```
public ImapClientException(
   string message,
   Exception innerException
);
```

Parameters

message

The error message that explains the reason for the exception.

innerException

The exception that is the cause of the current exception. If the ***innerException*** parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

ImapClientException Class | SocketTools Namespace | ImapClientException Constructor Overload List

ImapClientException Constructor (Int32)

Initializes a new instance of the ImapClientException class with a specified error number.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

[C#]

```
public ImapClientException(
   int code
);
```

Parameters

code

An integer value which specifies an error code.

Remarks

This constructor sets the **ErrorCode** property to the specified error code. For more information about the errors that may occur, refer to the ImapClient.ErrorCode enumeration.

See Also

ImapClientException Class | SocketTools Namespace | ImapClientException Constructor Overload List

ImapClientException Properties

The properties of the **ImapClientException** class are listed below. For a complete list of **ImapClientException** class members, see the ImapClientException Members topic.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

ImapClientException Class | SocketTools Namespace

ImapClientException.ErrorCode Property

Gets a value which specifies the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property ErrorCode As ErrorCode
```

[C#]

```
public ImapClient.ErrorCode ErrorCode {get;}
```

Property Value

Returns a ImapClient.ErrorCode enumeration value which specifies the error code.

Remarks

The **ErrorCode** property returns the error code that specifies the cause of the exception.

The default constructor for the ImapClientException class sets the error code to the last network error that occurred. For more information about the errors that may occur, refer to the ImapClient.ErrorCode enumeration.

See Also

ImapClientException Class | SocketTools Namespace

ImapClientException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]

```
Overrides Public ReadOnly Property Message As String
```

[C#]

```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

ImapClientException Class | SocketTools Namespace

ImapClientException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception. This value is the same as the values returned by the Windows Sockets API. For more information about socket error codes, see the Windows Socket Version 2 API error code documentation in MSDN.

See Also

ImapClientException Class | SocketTools Namespace

ImapClientException Methods

The methods of the **ImapClientException** class are listed below. For a complete list of **ImapClientException** class members, see the ImapClientException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

ImapClientException Class | SocketTools Namespace

ImapClientException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]
```
Overrides Public Function ToString() As String
```

[C#]
```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

ImapClientException Class | SocketTools Namespace

InternetDialer Class

Implements an interface to the Remote Access Services API.

For a list of all members of this type, see InternetDialer Members.

System.Object
  SocketTools.InternetDialer

[Visual Basic]
```
Public Class InternetDialer
    Implements IDisposable
```

[C#]
```
public class InternetDialer : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The InternetDialer class provides a way for client applications to connect to a remote server using Microsoft Windows Remote Access Services (RAS). To use this class, the dial-up networking software must be installed on the local system. For access to the Internet, the TCP/IP protocol must be installed and configured. The class may configured to use either the SLIP or PPP protocols, depending on the requirements of the service provider. Refer to your system documentation for information about installing and configuring dial-up networking on your system.

For those applications which may be used in a mobile environment, or otherwise require remote network access, the InternetDialer class provides a convenient interface to this service. Connections can be established and discontinued under the direct control of the program, rather than requiring that the user execute another program before starting your application.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetDialer (in SocketTools.InternetDialer.dll)

See Also

InternetDialer Members | SocketTools Namespace

InternetDialer Members

InternetDialer overview

Public Static (Shared) Fields

rasDeviceATM	A constant value which specifies an ATM device type.
rasDeviceFrameRelay	A constant value which specifies a frame relay device type.
rasDeviceGeneric	A constant value which specifies a generic device type.
rasDeviceIRDA	A constant value which specifies an infrared device type.
rasDeviceISDN	A constant value which specifies an ISDN device type.
rasDeviceModem	A constant value which specifies a modem device type.
rasDevicePad	A constant value which specifies a packet assembler/disassembler device type.
rasDeviceParallel	A constant value which specifies an parallel port device type.
rasDevicePPPoE	A constant value which specifies a PPP over Ethernet device type.
rasDeviceSerial	A constant value which specifies a serial port device type.
rasDeviceSonet	A constant value which specifies a SONET device type.
rasDeviceSW56	A constant value which specifies a SW56 device type.
rasDeviceVPN	A constant value which specifies a VPN device type.
rasDeviceX25	A constant value which specifies an X25 device type.

Public Instance Constructors

InternetDialer Constructor	Initializes a new instance of the InternetDialer class.

Public Instance Fields

Connection	Gets the handle for a dial-up networking session.
DeviceEntry	Gets the name of the specified device entry.
NameServer	Gets and sets the IP addresses of the nameservers assigned to the current phonebook entry.

PhoneBookEntry	Gets the name for the specified phone book entry.

Public Instance Properties

AreaCode	Gets and sets the area code for the current phonebook entry.
AutoConnect	Automatically inherit connections established by another process.
AutoDial	Enable and disable autodialing on the local system.
AutoDisconnect	Automatically disconnect from the remote server.
Blocking	Gets and sets the blocking state of the class.
BytesIn	Gets the number of bytes that have been received by the dial-up networking device.
BytesOut	Gets the number of bytes that have been transmitted by the dial-up networking device.
Callback	Specifies that the remote server should call the system back.
CallbackNumber	Gets and sets the telephone number for the remote server to call back on.
Connections	Gets the number of active dial-up networking sessions.
ConnectSpeed	Gets the line speed for the current dial-up networking connection.
CountryCode	Gets and sets the country code for the current phonebook entry.
CountryName	Gets and sets the country name for the current phonebook entry.
DefaultGateway	Enable and disable the default gateway for IP packets through the dial-up adapter.
DeviceCount	Gets the number of dial-up networking devices available.
DeviceName	Gets and sets the device name for the current dial-up networking connection.
DeviceType	Gets and sets the device type for the current dial-up networking connection.
DynamicAddress	Enables and disables the use of dynamically allocated IP addresses.
DynamicNameServers	Enables and disables the use of dynamically assigned nameserver addresses.
EntryName	Gets and sets the current phone book entry name.
FramingProtocol	Gets and sets the framing protocol for the current phonebook entry.

Handle	Gets and sets the handle for the current dial-up networking connection.
InternetAddress	Gets the address assigned to the current dial-up networking session.
Interval	Gets and sets the interval at which the connection is monitored.
IpHeaderCompression	Enables and disables IP header compression for the current phonebook entry.
IsConnected	Gets a value which indicates if a connection has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LcpExtensions	Enables and disables the use of PPP LCP extensions for the current phonebook entry.
LocalNumber	Gets and sets the local telephone number specified in the phonebook entry.
ModemLights	Enables and disables the dial-up networking system tray icon.
ModemSpeaker	Enables and disables the modem speaker.
NetworkLogon	Enables and disables a network login for the current phonebook entry.
NetworkProtocol	Gets and sets the network protocol for the current phonebook entry.
Password	Gets the password required to establish a connection with the service provider.
PhoneBook	Sets the file name of the Remote Access phonebook to use.
PhoneBookEntries	Gets the number of entries in the current phonebook.
PhoneNumber	Gets and sets the telephone number of the service provider.
RequireEncryption	Enables and disables secure authentication for the current phonebook entry.
ScriptFile	Gets and sets the name of the script file for the current phonebook entry.
ServerAddress	Gets the address of the dial-up networking server.
SoftwareCompression	Enables and disables software compression for the current phonebook entry.

Status	Gets a value which specifies the current status of the dial-up networking connection.
Terminal	Gets and sets the interactive terminal window mode for the current phonebook entry.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
UserDomain	Gets and sets the NT domain on which user authentication is to occur.
UserName	Gets the username required to establish a connection with the service provider.
UserPhoneBook	Gets the name of the default user phonebook.
Version	Gets a value which returns the current version of the InternetDialer class library.

Public Instance Methods

Connect	Overloaded. Establish a connection with a dial-up networking services provider.
CreateEntry	Create a new entry in the current phonebook.
DeleteEntry	Overloaded. Delete a phonebook entry from the current phonebook.
Disconnect	Terminate the connection with the dial-up networking service provider.
Dispose	Overloaded. Releases all resources used by InternetDialer.
EditEntry	Edit an existing phonebook entry in the current phonebook.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the InternetDialer class.
LoadEntry	Overloaded. Load the specified entry from the current phonebook.
RenameEntry	Rename an existing phonebook entry.
Reset	Reset the internal state of the object, resetting all properties to their default values.

SaveEntry	Overloaded. Save the current settings to the specified phonebook entry in the current phonebook.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.

Public Instance Events

OnCancel	Occurs when a blocking client operation is canceled.
OnConnect	Occurs when a connection is established with the service provider.
OnDisconnect	Occurs when the dial-up networking connection is terminated.
OnError	Occurs when an client operation fails.
OnStatus	Occurs when the when the connection state changes.
OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the InternetDialer class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer Constructor

Initializes a new instance of the InternetDialer class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public InternetDialer();
```

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer Fields

The fields of the **InternetDialer** class are listed below. For a complete list of **InternetDialer** class members, see the InternetDialer Members topic.

Public Static (Shared) Fields

rasDeviceATM	A constant value which specifies an ATM device type.
rasDeviceFrameRelay	A constant value which specifies a frame relay device type.
rasDeviceGeneric	A constant value which specifies a generic device type.
rasDeviceIRDA	A constant value which specifies an infrared device type.
rasDeviceISDN	A constant value which specifies an ISDN device type.
rasDeviceModem	A constant value which specifies a modem device type.
rasDevicePad	A constant value which specifies a packet assembler/disassembler device type.
rasDeviceParallel	A constant value which specifies an parallel port device type.
rasDevicePPPoE	A constant value which specifies a PPP over Ethernet device type.
rasDeviceSerial	A constant value which specifies a serial port device type.
rasDeviceSonet	A constant value which specifies a SONET device type.
rasDeviceSW56	A constant value which specifies a SW56 device type.
rasDeviceVPN	A constant value which specifies a VPN device type.
rasDeviceX25	A constant value which specifies an X25 device type.

Public Instance Fields

Connection	Gets the handle for a dial-up networking session.
DeviceEntry	Gets the name of the specified device entry.
NameServer	Gets and sets the IP addresses of the nameservers assigned to the current phonebook entry.
PhoneBookEntry	Gets the name for the specified phone book entry.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.Connection Field

Gets the handle for a dial-up networking session.

[Visual Basic]

```
Public ReadOnly Connection As ConnectionArray
```

[C#]

```
public readonly ConnectionArray Connection;
```

Remarks

The **Connection** array can be used to enumerate the active dial-up networking sessions on the local system. The index is zero-based, and the number of connections is returned by the **Connections** property. The property returns an integer value which represents the handle to the session. Setting the **Handle** property to this value will cause the control to inherit the session and the control's properties will be updated with information about the connection.

Specifying an index greater than the number of available connections will generate an exception.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.DeviceEntry Field

Gets the name of the specified device entry.

[Visual Basic]

```
Public ReadOnly DeviceEntry As DeviceEntryArray
```

[C#]

```
public readonly DeviceEntryArray DeviceEntry;
```

Remarks

The **DeviceEntry** array can be used in conjunction with the **DeviceCount** property to enumerate the available dial-up networking devices. Typically this is used to provide a user with a selection of dial-up devices. The device used by the current phonebook entry can be changed by setting the **DeviceName** property to one of the device entry values.

Note that you should first set the **DeviceType** property to the type of device which you wish to enumerate. The default device type is **rasDeviceModem**, for serial analog modems or other devices which recognize the AT command set.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.NameServer Field

Gets and sets the IP addresses of the nameservers assigned to the current phonebook entry.

[Visual Basic]
```
Public ReadOnly NameServer As NameServerArray
```

[C#]
```
public readonly NameServerArray NameServer;
```

Remarks

The **NameServer** array is used to set or return the nameserver IP addresses assigned to the current phonebook entry. Setting the array to an IP address changes the corresponding address assigned to the phonebook entry. Note that assigned nameserver addresses are only used if the **DynamicNameServers** property has been set to **false**. If dynamic nameservers are assigned to the session this array will not return those addresses, it will return empty strings.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.PhoneBookEntry Field

Gets the name for the specified phone book entry.

[Visual Basic]

```
Public ReadOnly PhoneBookEntry As PhoneBookEntryArray
```

[C#]

```
public readonly PhoneBookEntryArray PhoneBookEntry;
```

Remarks

The **PhoneBookEntry** array contains a list of the entries in the current phone book, and may be used to establish a connection with a remote server. Specifying an index greater than the number of available entries in the phone book will generate an exception.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer Properties

The properties of the **InternetDialer** class are listed below. For a complete list of **InternetDialer** class members, see the InternetDialer Members topic.

Public Instance Properties

AreaCode	Gets and sets the area code for the current phonebook entry.
AutoConnect	Automatically inherit connections established by another process.
AutoDial	Enable and disable autodialing on the local system.
AutoDisconnect	Automatically disconnect from the remote server.
Blocking	Gets and sets the blocking state of the class.
BytesIn	Gets the number of bytes that have been received by the dial-up networking device.
BytesOut	Gets the number of bytes that have been transmitted by the dial-up networking device.
Callback	Specifies that the remote server should call the system back.
CallbackNumber	Gets and sets the telephone number for the remote server to call back on.
Connections	Gets the number of active dial-up networking sessions.
ConnectSpeed	Gets the line speed for the current dial-up networking connection.
CountryCode	Gets and sets the country code for the current phonebook entry.
CountryName	Gets and sets the country name for the current phonebook entry.
DefaultGateway	Enable and disable the default gateway for IP packets through the dial-up adapter.
DeviceCount	Gets the number of dial-up networking devices available.
DeviceName	Gets and sets the device name for the current dial-up networking connection.
DeviceType	Gets and sets the device type for the current dial-up networking connection.
DynamicAddress	Enables and disables the use of dynamically allocated IP addresses.
DynamicNameServers	Enables and disables the use of dynamically assigned nameserver addresses.

EntryName	Gets and sets the current phone book entry name.
FramingProtocol	Gets and sets the framing protocol for the current phonebook entry.
Handle	Gets and sets the handle for the current dial-up networking connection.
InternetAddress	Gets the address assigned to the current dial-up networking session.
Interval	Gets and sets the interval at which the connection is monitored.
IpHeaderCompression	Enables and disables IP header compression for the current phonebook entry.
IsConnected	Gets a value which indicates if a connection has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LcpExtensions	Enables and disables the use of PPP LCP extensions for the current phonebook entry.
LocalNumber	Gets and sets the local telephone number specified in the phonebook entry.
ModemLights	Enables and disables the dial-up networking system tray icon.
ModemSpeaker	Enables and disables the modem speaker.
NetworkLogon	Enables and disables a network login for the current phonebook entry.
NetworkProtocol	Gets and sets the network protocol for the current phonebook entry.
Password	Gets the password required to establish a connection with the service provider.
PhoneBook	Sets the file name of the Remote Access phonebook to use.
PhoneBookEntries	Gets the number of entries in the current phonebook.
PhoneNumber	Gets and sets the telephone number of the service provider.
RequireEncryption	Enables and disables secure authentication for the current phonebook entry.
ScriptFile	Gets and sets the name of the script file for the current phonebook entry.

ServerAddress	Gets the address of the dial-up networking server.
SoftwareCompression	Enables and disables software compression for the current phonebook entry.
Status	Gets a value which specifies the current status of the dial-up networking connection.
Terminal	Gets and sets the interactive terminal window mode for the current phonebook entry.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
UserDomain	Gets and sets the NT domain on which user authentication is to occur.
UserName	Gets the username required to establish a connection with the service provider.
UserPhoneBook	Gets the name of the default user phonebook.
Version	Gets a value which returns the current version of the InternetDialer class library.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.AreaCode Property

Gets and sets the area code for the current phonebook entry.

[Visual Basic]
```
Public Property AreaCode As String
```

[C#]
```
public string AreaCode {get; set;}
```

Property Value

A string which specifies the area code.

Remarks

The **AreaCode** property is used to set or return the current phonebook entry's area code. If no area code has been specified, then this property will return an empty string. The value of this property is ignored unless the **CountryCode** property is also set to a valid country code.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.AutoConnect Property

Automatically inherit connections established by another process.

[Visual Basic]

```
Public Property AutoConnect As Boolean
```

[C#]

```
public bool AutoConnect {get; set;}
```

Property Value

A boolean value which specifies if connections are automatically inherited by the class.

Remarks

The **AutoConnect** property determines if the class automatically detects if a connection has been established by another process. When enabled, the class will periodically check for any connections that have been established. The **Interval** property controls the frequency in which the control performs this check.

If the class detects that a connection has been made, it will immediately fire the **OnConnect** event, followed by the **OnStatus** event, to indicate that a connection has been established. The class then begins to monitor that connection as usual, until that connection is dropped or the control is unloaded.

To periodically check to see if a connection has been established by another process without using the **AutoConnect** property, read the value of the **Connections** property, which returns the number of active dial-up networking connections. A value greater than zero indicates that a dial-up networking connection has been established.

If there are multiple dial-up networking devices on the system, it may be possible for more than one connection to be active at a time. If this is the case, setting the **AutoConnect** property to **true** will cause the class to inherit the first active connection. To manage multiple dial-up connections, use the **Connection** array to enumerate the available connections and set the **Handle** property to take control of a specific session.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.AutoDial Property

Enable and disable autodialing on the local system.

[Visual Basic]

```
Public Property AutoDial As Boolean
```

[C#]

```
public bool AutoDial {get; set;}
```

Property Value

A boolean value which specifies if autodialing has been enabled.

Remarks

The **AutoDial** property can be used to determine if autodialing is enabled or disabled on the current system. When autodialing is enabled and an application attempts to establish a connection over the Internet, a dialog box will be displayed asking the user if they want to connect to their default service provider. This property will return **true** if autodialing is currently enabled, or **false** if it has been disabled.

Setting the **AutoDial** property allows an application to change the autodial settings for the current user. Setting the property value to **true** specifies that you wish to enable autodialing, and the system will prompt the user to establish a dial-up connection when necessary. Setting the property to **false** disables autodialing, and prevents the system from prompting the user. This can be beneficial if your application needs to run in an unattended mode. If you change the autodial settings for the user, it is recommended that you restore them to their original value before the application terminates.

If the autodial settings cannot be changed by the current user, an exception will be generated.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.AutoDisconnect Property

Automatically disconnect from the remote server.

[Visual Basic]

```
Public Property AutoDisconnect As Boolean
```

[C#]

```
public bool AutoDisconnect {get; set;}
```

Property Value

A boolean value that specifies if connections are automatically terminated.

Remarks

The **AutoDisconnect** property determines if this instance of the class should automatically disconnect from a remote host when the destructor is called, typically when the application terminates. The default value for this property is **true**.

If a dial-up connection was already established at the time an instance of the class is created, this property will be reset to **false**, preventing it from automatically disconnecting from the host when it is unloaded. Therefore, to always force the control to automatically terminate a connection when it is unloaded, you must explicitly set the property value to **true** in your application.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.Blocking Property

Gets and sets the blocking state of the class.

[Visual Basic]

```
Public Property Blocking As Boolean
```

[C#]

```
public bool Blocking {get; set;}
```

Property Value

A boolean value which specifies the blocking state of the class.

Remarks

The **Blocking** property determines how the class establishes a dial-up connection. If set to **true**, the class will wait until a connection has been established or the connection attempt fails before returning control to the application. If set to **false**, the class will begin the connection process and return control immediately to the application. For a non-blocking connection, the application should monitor the **OnStatus** event to determine the progress of the connection attempt. The default value for this property is **false**.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.BytesIn Property

Gets the number of bytes that have been received by the dial-up networking device.

[Visual Basic]

```
Public ReadOnly Property BytesIn As Integer
```

[C#]

```
public int BytesIn {get;}
```

Property Value

An integer which specifies the number of bytes received.

Remarks

The **BytesIn** property returns the number of bytes that have been received by the dial-up networking device. If the control is unable to determine the number of bytes received, it will return a value of zero.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.BytesOut Property

Gets the number of bytes that have been transmitted by the dial-up networking device.

[Visual Basic]

```
Public ReadOnly Property BytesOut As Integer
```

[C#]

```
public int BytesOut {get;}
```

Property Value

An integer which specifies the number of bytes transmitted.

Remarks

The **BytesOut** property returns the number of bytes that have been transmitted by the dial-up networking device. If the control is unable to determine the number of bytes transmitted, it will return a value of zero.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.Callback Property

Specifies that the remote server should call the system back.

[Visual Basic]

```
Public Property Callback As Boolean
```

[C#]

```
public bool Callback {get; set;}
```

Property Value

A boolean value which specifies if the server should call the local system back.

Remarks

Setting the **Callback** property specifies that the server should call the user back at the telephone number specified by the **CallbackNumber** property. This property is ignored unless the user has "Set By Caller" callback permission on the server.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.CallbackNumber Property

Gets and sets the telephone number for the remote server to call back on.

[Visual Basic]

```
Public Property CallbackNumber As String
```

[C#]

```
public string CallbackNumber {get; set;}
```

Property Value

A string which specifies the callback telephone number.

Remarks

Setting the **CallbackNumber** property specifies that the server should call the user back at the given telephone number. This property is ignored unless the user has "Set By Caller" callback permission on the server. Assigning an asterisk to this property causes the number stored in the phone book entry to be used for callback.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.Connections Property

Gets the number of active dial-up networking sessions.

[Visual Basic]

```
Public ReadOnly Property Connections As Integer
```

[C#]

```
public int Connections {get;}
```

Property Value

An integer value which specifies the number of active dial-up networking sessions.

Remarks

The **Connections** property returns the number of active dial-up networking connections on the local system. A value of zero indicates that there is no dial-up networking connection. This property is used in conjunction with the **Connection** array to enumerate the connections on the current system.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.ConnectSpeed Property

Gets the line speed for the current dial-up networking connection.

[Visual Basic]
```
Public ReadOnly Property ConnectSpeed As Integer
```

[C#]
```
public int ConnectSpeed {get;}
```

Property Value

An integer value which specifies the connection speed.

Remarks

The **ConnectSpeed** property returns the speed, in bytes per second, at which the current dial-up networking device has established a connection. If the class is unable to determine the connection speed, it will return a value of zero.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.CountryCode Property

Gets and sets the country code for the current phonebook entry.

[Visual Basic]

```
Public Property CountryCode As Integer
```

[C#]

```
public int CountryCode {get; set;}
```

Property Value

An integer value which specifies the country code.

Remarks

The **CountryCode** property specifies the numeric country code for the current phonebook entry. If this value is zero, then the country and area code information is not used when dialing the phone number. The country code for the United States is 1.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.CountryName Property

Gets and sets the country name for the current phonebook entry.

[Visual Basic]

```
Public Property CountryName As String
```

[C#]

```
public string CountryName {get; set;}
```

Property Value

A string which specifies the country name.

Remarks

The **CountryName** property returns the name of the country associated with the country code used when dialing the current phonebook entry. If no country code has been specified, this property will return an empty string. Setting this property to the name of a country will change the current country code. If no area code has been defined, and the country code specifies the current dialing location, the **AreaCode** property will be updated to the current area code.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.DefaultGateway Property

Enable and disable the default gateway for IP packets through the dial-up adapter.

[Visual Basic]

```
Public Property DefaultGateway As Boolean
```

[C#]

```
public bool DefaultGateway {get; set;}
```

Property Value

A boolean value which specifies if the default gateway should be used.

Remarks

The **DefaultGateway** property is used to determine the default gateway for IP packets. If set to **true**, then packets are routed through the dial-up networking adapter when the connection is active. The value of this property corresponds to the Use Default Gateway checkbox on the TCP/IP configuration dialog.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.DeviceCount Property

Gets the number of dial-up networking devices available.

[Visual Basic]

```
Public ReadOnly Property DeviceCount As Integer
```

[C#]

```
public int DeviceCount {get;}
```

Property Value

An integer value which specifies the number of devices.

Remarks

The **DeviceCount** property returns the number of dial-up networking devices available. This property can be used in conjunction with the **DeviceEntry** array to enumerate the devices.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.DeviceName Property

Gets and sets the device name for the current dial-up networking connection.

[Visual Basic]

```
Public Property DeviceName As String
```

[C#]

```
public string DeviceName {get; set;}
```

Property Value

A string which specifies the device name.

Remarks

The **DeviceName** property returns a description of the device that the connection was established on. For example, the string "US Robotics Sportster 56000" may be returned for a modem. Note that this property value may change if the **DeviceType** property is modified. Setting this property will change the device used to establish the dial-up networking connection. Changes to this property value should be made after changes to the **DeviceType** property.

To enumerate a list of available devices for a given device type, use the **DeviceCount** property and **DeviceEntry** array.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.DeviceType Property

Gets and sets the device type for the current dial-up networking connection.

[Visual Basic]

```
Public Property DeviceType As String
```

[C#]

```
public string DeviceType {get; set;}
```

Property Value

A string which specifies the device type.

Remarks

The **DeviceType** property returns the type of device that the connection was established with. Setting this property will change the type of device that will be used to establish the connection. Examples of valid device types are:

DeviceType	Description
modem	An internal or external serial analog modem device, or other serial communications device which supports the AT command set.
isdn	An ISDN terminal adapter. Note that some ISDN devices, such as the 3Com ImpactIQ are considered modem devices.
vpn	A virtual private network connection.

Because changing the device type can change the current device name, it is recommended that applications change this property value before changing the value of the **DeviceName** property.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.DynamicAddress Property

Enables and disables the use of dynamically allocated IP addresses.

[Visual Basic]

```
Public Property DynamicAddress As Boolean
```

[C#]

```
public bool DynamicAddress {get; set;}
```

Property Value

A boolean value which specifies if a dynamically allocated IP address should be used.

Remarks

The **DynamicAddress** property determines if the current phonebook entry should use a dynamically assigned IP address. If this property is set to **true**, then an IP address is assigned to the dial-up adapter when the connection is established. If set to **false**, then the dial-up adapter IP address is set to the value of the **InternetAddress** property.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.DynamicNameServers Property

Enables and disables the use of dynamically assigned nameserver addresses.

[Visual Basic]

```
Public Property DynamicNameServers As Boolean
```

[C#]

```
public bool DynamicNameServers {get; set;}
```

Property Value

A boolean value which specifies if dynamically allocated nameservers should be used.

Remarks

The **DynamicNameServers** property determines if the current phonebook entry should use dynamically assigned nameservers. If this property is set to **true**, then one or more nameservers are assigned to the dial-up adapter when the connection is established. If set to **false**, then the dial-up adapter nameservers are set to the values specified by the **NameServer** property array.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.EntryName Property

Gets and sets the current phone book entry name.

[Visual Basic]

```
Public Property EntryName As String
```

[C#]

```
public string EntryName {get; set;}
```

Property Value

A string which specifies the current phonebook entry name.

Remarks

The **EntryName** property can be used to specify a phone book entry to use to connect with a remote server. The entry name identifies a communications profile which includes the telephone number, callback number and domain name of the remote host. Setting the **EntryName** property to an empty string indicates that a telephone number will be provided to establish the connection.

In Windows documentation, the phonebook entry name is also referred to as a ***connectoid***.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.FramingProtocol Property

Gets and sets the framing protocol for the current phonebook entry.

[Visual Basic]

```
Public Property FramingProtocol As RasFramingProtocol
```

[C#]

```
public InternetDialer.RasFramingProtocol FramingProtocol {get; set;}
```

Property Value

A RasFramingProtocol enumeration which specifies the framing protocol.

Remarks

The **FramingProtocol** property is used to set or return the framing protocol used for the current phonebook entry.

Note that unless there is a specific need for the application to use SLIP or the Microsoft RAS protocol, it is recommended that PPP always be selected as the framing protocol.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.Handle Property

Gets and sets the handle for the current dial-up networking connection.

[Visual Basic]

```
Public Property Handle As Integer
```

[C#]

```
public int Handle {get; set;}
```

Property Value

An integer value which specifies a dial-up networking connection.

Remarks

The **Handle** property returns the handle to the current dial-up networking connection, or a value of zero if the class has not been used to establish a connection. Setting the value of this property to a valid handle causes the class to inherit the specified connection, and its properties will be updated with information about that connection. This enables an application to monitor and control a connection that was established by another program.

Setting the **Handle** property to a value of zero causes the class to release the current connection, however it will not cause the dial-up networking session to terminate. To disconnect from the remote server, the **Disconnect** method must be called by the application. Setting the property to a non-zero value which does not specify a valid handle will generate an exception.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.InternetAddress Property

Gets the address assigned to the current dial-up networking session.

[Visual Basic]
```
Public Property InternetAddress As String
```

[C#]
```
public string InternetAddress {get; set;}
```

Property Value

A string which specifies an Internet Protocol (IP) address.

Remarks

The **InternetAddress** property returns the address assigned to the current dial-up networking session. If no connection has been established, or the connection has not been made with a PPP server, then this property will return an empty string. If the **DynamicAddress** property is set to **false**, changing this property value will update the address assigned to the current phonebook entry.

The address may only be changed before a connection is established.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.Interval Property

Gets and sets the interval at which the connection is monitored.

[Visual Basic]

```
Public Property Interval As Integer
```

[C#]

```
public int Interval {get; set;}
```

Property Value

An integer value which specifies the interval in milliseconds.

Remarks

The **Interval** property specifies the interval, in milliseconds, at which the connection is monitored by the class. The minimum value of 0 indicates that the class should not monitor the connection. The maximum interval value is 65536 milliseconds, which is slightly more than one minute. The default value is 1000, which causes the control to check the connection status every second.

Note that setting the property value to zero will prevent the class from detecting certain conditions, such as a disconnected telephone line or a modem that is turned off.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.IpHeaderCompression Property

Enables and disables IP header compression for the current phonebook entry.

[Visual Basic]

```
Public Property IpHeaderCompression As Boolean
```

[C#]

```
public bool IpHeaderCompression {get; set;}
```

Property Value

A boolean value which specifies if IP header compression is enabled.

Remarks

The **IpHeaderCompression** property is used to enable or disable IP header compression. If set to **true**, when a connection is established, RAS will negotiate with the dial-up server to use header compression. If set to **false**, header compression will not be negotiated. This property corresponds to the Use IP Header Compression checkbox on the TCP/IP configuration dialog.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.IsConnected Property

Gets a value which indicates if a connection has been established.

[Visual Basic]

```
Public ReadOnly Property IsConnected As Boolean
```

[C#]

```
public bool IsConnected {get;}
```

Property Value

A boolean value which specifies if a connection has been established with a service provider.

Remarks

The **IsConnected** property is used to determine if the class has connected to the remote host. A value of **true** indicates that a connection has been established.

Note that the **IsConnected** property should not be used to determine if an active dial-up networking connection has been established by another application. The property will only return **true** if the class has been used to establish the connection itself, or if a connection is inherited by setting either the **AutoConnect** or **Handle** properties. To determine if there are any active dial-up networking connections, check the value of the **Connections** property.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.IsInitialized Property

Gets a value which indicates if the current instance of the class has been initialized successfully.

[Visual Basic]

```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]

```
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the class instance has been initialized; otherwise returns **false**.

Remarks

The **IsInitialized** property is used to determine if the current instance of the class has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the class may not be able to initialize itself.

The most common reasons that a class instance may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required networking libraries or not being able to access the system registry.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public InternetDialer.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.LcpExtensions Property

Enables and disables the use of PPP LCP extensions for the current phonebook entry.

[Visual Basic]

```
Public Property LcpExtensions As Boolean
```

[C#]

```
public bool LcpExtensions {get; set;}
```

Property Value

A boolean value which specifies if PPP LCP extensions are enabled.

Remarks

The **LcpExtensions** property determines if the PPP LCP extensions defined in RFC 1570 will be used. If the PPP framing protocol is being used for the dial-up connection, it is recommended that this property be set to **true**. However, some older implementations of PPP may require that this property be set to **false** in order to establish a connection.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.LocalNumber Property

Gets and sets the local telephone number specified in the phonebook entry.

[Visual Basic]

```
Public Property LocalNumber As String
```

[C#]

```
public string LocalNumber {get; set;}
```

Property Value

A string which specifies the local telephone number.

Remarks

The **LocalNumber** property sets or returns the local phone number that is specified in the current phonebook entry. If the **CountryCode** property has a value of zero, then the local number is dialed to connect to the server. If the **CountryCode** property is set to a valid country code, then RAS will also use the country and area code values when dialing the phone number.

Note that this property only determines the local phone number for the phonebook entry, and can be overridden by setting the **PhoneNumber** property to a specific value.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.ModemLights Property

Enables and disables the dial-up networking system tray icon.

[Visual Basic]

```
Public Property ModemLights As Boolean
```

[C#]

```
public bool ModemLights {get; set;}
```

Property Value

A boolean value which specifies if the system tray icon is displayed.

Remarks

The **ModemLights** property determines if the dial-up networking icon in the system tray is displayed when a connection is established.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.ModemSpeaker Property

Enables and disables the modem speaker.

[Visual Basic]

```
Public Property ModemSpeaker As Boolean
```

[C#]

```
public bool ModemSpeaker {get; set;}
```

Property Value

A boolean value which specifies if the modem speaker is enabled.

Remarks

The **ModemSpeaker** property determines if the modem speaker is enabled when dialing the remote server. If the property is set to **false**, the modem will be silent when dialing the telephone number and establishing the connection. Note that setting this property to **true** will not force the speaker on if the modem hardware has been configured to explicitly disable the speaker.

To disable the speaker, the modem must support changes to the speaker volume. Disabling the speaker is typically done by instructing the modem to set the speaker volume to zero.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.NetworkLogon Property

Enables and disables a network login for the current phonebook entry.

[Visual Basic]

```
Public Property NetworkLogon As Boolean
```

[C#]

```
public bool NetworkLogon {get; set;}
```

Property Value

A boolean value which specifies if a network login is enabled.

Remarks

The **NetworkLogon** property determines if the client automatically logs on to the network after a connection has been established.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.NetworkProtocol Property

Gets and sets the network protocol for the current phonebook entry.

[Visual Basic]

```
Public Property NetworkProtocol As RasNetworkProtocol
```

[C#]

```
public InternetDialer.RasNetworkProtocol NetworkProtocol {get; set;}
```

Property Value

A RasNetworkProtocol enumeration value which specifies the network protocol.

Remarks

The **NetworkProtocol** property is used to set or return the network protocol used for the current phonebook entry.

Note that unless there is a specific need for the application to use the NetBEUI or IPX protocols, it is recommended that only the TCP/IP protocol be specified.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.Password Property

Gets the password required to establish a connection with the service provider.

[Visual Basic]

```
Public Property Password As String
```

[C#]

```
public string Password {get; set;}
```

Property Value

A string which specifies the password for the current phonebook entry.

Remarks

The **Password** property specifies the password required to establish a connection with the service provider.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.PhoneBook Property

Sets the file name of the Remote Access phonebook to use.

[Visual Basic]

```
Public Property PhoneBook As String
```

[C#]

```
public string PhoneBook {get; set;}
```

Property Value

A string which specifies the current phonebook.

Remarks

The **PhoneBook** property specifies the file name of the Remote Access phone book. Setting this property to an empty string causes the default phonebook to be used.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.PhoneBookEntries Property

Gets the number of entries in the current phonebook.

[Visual Basic]
```
Public ReadOnly Property PhoneBookEntries As Integer
```

[C#]
```
public int PhoneBookEntries {get;}
```

Property Value

An integer value which specifies the number of phonebook entries.

Remarks

The **PhoneBookEntries** property returns the number of entries in the current phonebook. A value of zero indicates that no phonebook entries are available.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.PhoneNumber Property

Gets and sets the telephone number of the service provider.

[Visual Basic]

```
Public Property PhoneNumber As String
```

[C#]

```
public string PhoneNumber {get; set;}
```

Property Value

A string which specifies a telephone number.

Remarks

The **PhoneNumber** property specifies the telephone number of the service provider. If this property is not set, then the **PhoneEntry** property must be set to a valid phone book entry. If both the **PhoneNumber** and **PhoneEntry** properties are defined, the **PhoneNumber** property will override the value specified in the phonebook.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.RequireEncryption Property

Enables and disables secure authentication for the current phonebook entry.

[Visual Basic]
```
Public Property RequireEncryption As Boolean
```

[C#]
```
public bool RequireEncryption {get; set;}
```

Property Value

A boolean value which specifies if secure authentication is enabled.

Remarks

The **RequireEncryption** property determines if encryption is required during PPP authentication. If the property is set to **true**, then only secure password schemes can be used to authenticate the client. If the property is set to **false**, the client can use the PAP plain-text authentication protocol to authenticate the client. Some older PPP implementations may require that this property be set to **false** in order to establish a connection.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.ScriptFile Property

Gets and sets the name of the script file for the current phonebook entry.

[Visual Basic]

```
Public Property ScriptFile As String
```

[C#]

```
public string ScriptFile {get; set;}
```

Property Value

A string which specifies the name of the script file.

Remarks

The **ScriptFile** property specifies the name of the login script used to establish a connection with the remote host. This property must be set to the full pathname of the script file. If a script file is not required, then this property should be set to an empty string.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.ServerAddress Property

Gets the address of the dial-up networking server.

[Visual Basic]

```
Public ReadOnly Property ServerAddress As String
```

[C#]

```
public string ServerAddress {get;}
```

Property Value

A string which specifies an Internet Protocol (IP) address.

Remarks

The **ServerAddress** property returns the address of the dial-up networking server that the local host has connected to. If no connection has been established, or the connection has not been made with a PPP server, then this property will return an empty string. This property may also return an empty string if the remote server did not provide this information during the connection process.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.SoftwareCompression Property

Enables and disables software compression for the current phonebook entry.

[Visual Basic]

```
Public Property SoftwareCompression As Boolean
```

[C#]

```
public bool SoftwareCompression {get; set;}
```

Property Value

A boolean value which specifies if software compression is enabled.

Remarks

The **SoftwareCompression** property determines if data compression is negotiated during the connection. If the property is set to **true**, then the client will negotiate a compatible compression protocol. Software compression should only be disabled if the client is unable to establish a connection with the server.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.Status Property

Gets a value which specifies the current status of the dial-up networking connection.

[Visual Basic]

```
Public ReadOnly Property Status As DialerStatus
```

[C#]

```
public InternetDialer.DialerStatus Status {get;}
```

Property Value

A DialerStatus enumeration value which specifies the current status.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.Terminal Property

Gets and sets the interactive terminal window mode for the current phonebook entry.

[Visual Basic]
```
Public Property Terminal As RasTerminalMode
```

[C#]
```
public InternetDialer.RasTerminalMode Terminal {get; set;}
```

Property Value

A RasTerminalMode enumeration which specifies the terminal window mode.

Remarks

The **Terminal** array is used to control if a terminal window is displayed during the dial-up networking connection process. The terminal window can be used to allow user input before and/or after the dial-up networking connection has been established. If scripting has been enabled by setting the **ScriptFile** property, no terminal window should be displayed after the connection. This is because scripting has it's own terminal implementation.

Displaying a terminal window also imposes several restrictions on the behavior of the class. Because of how the Remote Access Services API is implemented by Microsoft, a connection dialog will be displayed after the **Connect** method is called if the **Terminal** property is non-zero. Setting this property to a non-zero value will also disable any asynchronous event notifications. It is not recommended that you set this property unless it is absolutely necessary.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]
```
Public Property ThrowError As Boolean
```

[C#]
```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.Timeout Property

Gets and sets a value which specifies a timeout period in seconds.

[Visual Basic]

```
Public Property Timeout As Integer
```

[C#]

```
public int Timeout {get; set;}
```

Property Value

An integer value which specifies a timeout period in seconds.

Remarks

Setting the **Timeout** property specifies the number of seconds until a blocking operation fails and returns an error. The timeout period is only used when the client is in blocking mode. Although this property can be changed when the client is in non-blocking mode, the value will be ignored until the client is returned to blocking mode.

For most applications it is recommended the timeout period be set between 10 and 20 seconds.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.UserDomain Property

Gets and sets the NT domain on which user authentication is to occur.

[Visual Basic]
```
Public Property UserDomain As String
```

[C#]
```
public string UserDomain {get; set;}
```

Property Value

A string which specifies the NT domain name.

Remarks

The **UserDomain** property is used to specify the NT domain on which the user name and password will be authenticated. An empty string specifies the domain in which the Remote Access server is a member. An asterisk specifies the domain stored in the phone book entry.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.UserName Property

Gets the username required to establish a connection with the service provider.

[Visual Basic]

```
Public Property UserName As String
```

[C#]

```
public string UserName {get; set;}
```

Property Value

A string which specifies the username for the current phonebook entry.

Remarks

The **UserName** property specifies the username required to establish a connection with the service provider.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.UserPhoneBook Property

Gets the name of the default user phonebook.

[Visual Basic]

```
Public ReadOnly Property UserPhoneBook As String
```

[C#]

```
public string UserPhoneBook {get;}
```

Property Value

A string which specifies a phonebook name.

Remarks

The **UserPhoneBook** property returns the name of the default user phonebook. The value returned depends on how the user has configured dial-up networking, specifically whether the system, user or alternate phonebook has been selected. The current phonebook can be changed by setting the **PhoneBook** property.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.Version Property

Gets a value which returns the current version of the InternetDialer class library.

[Visual Basic]

```
Public ReadOnly Property Version As String
```

[C#]

```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the InternetDialer class library. This value can be used by an application for validation and debugging purposes.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer Methods

The methods of the **InternetDialer** class are listed below. For a complete list of **InternetDialer** class members, see the InternetDialer Members topic.

Public Instance Methods

Connect	Overloaded. Establish a connection with a dial-up networking services provider.
CreateEntry	Create a new entry in the current phonebook.
DeleteEntry	Overloaded. Delete a phonebook entry from the current phonebook.
Disconnect	Terminate the connection with the dial-up networking service provider.
Dispose	Overloaded. Releases all resources used by InternetDialer.
EditEntry	Edit an existing phonebook entry in the current phonebook.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the InternetDialer class.
LoadEntry	Overloaded. Load the specified entry from the current phonebook.
RenameEntry	Rename an existing phonebook entry.
Reset	Reset the internal state of the object, resetting all properties to their default values.
SaveEntry	Overloaded. Save the current settings to the specified phonebook entry in the current phonebook.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the InternetDialer class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading

	the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.Connect Method

Establish a connection with a dial-up networking services provider.

Overload List

Establish a connection with a dial-up networking services provider.

public bool Connect();

Establish a connection with a dial-up networking services provider.

public bool Connect(string);

Establish a connection with a dial-up networking services provider.

public bool Connect(string,string,string);

Establish a connection with a dial-up networking services provider.

public bool Connect(string,string,string,string);

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.Connect Method ()

Establish a connection with a dial-up networking services provider.

[Visual Basic]

```
Overloads Public Function Connect() As Boolean
```

[C#]

```
public bool Connect();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Connect** method establishes a dial-up networking connection with a service provider using the current phonebook entry.

The current phonebook entry name is specified by the **EntryName** property.

See Also

InternetDialer Class | SocketTools Namespace | InternetDialer.Connect Overload List

InternetDialer.Connect Method (String)

Establish a connection with a dial-up networking services provider.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal entryName As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string entryName
);
```

Parameters

entryName

A string which specifies the phonebook entry that should be used when establishing the connection.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Connect** method establishes a dial-up networking connection with a service provider using the specified phonebook entry. The entry name is the same name as the connectoid that is displayed when you list the available dial-up networking connections on the local system.

For a list of all of the available phonebook entries, reference the **PhoneBookEntry** array.

See Also

InternetDialer Class | SocketTools Namespace | InternetDialer.Connect Overload List

InternetDialer.Connect Method (String, String, String)

Establish a connection with a dial-up networking services provider.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal phoneNumber As String, _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string phoneNumber,
   string userName,
   string userPassword
);
```

Parameters

phoneNumber
A string which specifies the phone number to dial.

userName
A string which specifies the username which will be used to authenticate the session.

userPassword
A string which specifies the password which will be used to authenticate the session.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Connect** method establishes a dial-up networking connection with a service provider. A temporary phonebook entry will be created for the dial-up networking session, and this entry will be removed when the local host disconnects from the service provider.

See Also

InternetDialer Class | SocketTools Namespace | InternetDialer.Connect Overload List

InternetDialer.Connect Method (String, String, String, String)

Establish a connection with a dial-up networking services provider.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal phoneNumber As String, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal userDomain As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string phoneNumber,
   string userName,
   string userPassword,
   string userDomain
);
```

Parameters

phoneNumber
A string which specifies the phone number to dial.

userName
A string which specifies the username which will be used to authenticate the session.

userPassword
A string which specifies the password which will be used to authenticate the session.

userDomain
A string which specifies the domain on which the username and password will be authenticated. An empty string specifies the domain in which the Remote Access Server is a member.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Connect** method establishes a dial-up networking connection with a service provider. A temporary phonebook entry will be created for the dial-up networking session, and this entry will be removed when the local host disconnects from the service provider.

See Also

InternetDialer Class | SocketTools Namespace | InternetDialer.Connect Overload List

InternetDialer.CreateEntry Method

Create a new entry in the current phonebook.

[Visual Basic]
```
Public Function CreateEntry() As Boolean
```

[C#]
```
public bool CreateEntry();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateEntry** method displays a dialog box which allows the user to create a new phonebook entry on the system. If you do not wish to display a dialog box, use the **SaveEntry** method instead.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.DeleteEntry Method

Delete the current phonebook entry from the phonebook.

Overload List

Delete the current phonebook entry from the phonebook.

public bool DeleteEntry();

Delete a phonebook entry from the current phonebook.

public bool DeleteEntry(string);

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.DeleteEntry Method ()

Delete the current phonebook entry from the phonebook.

[Visual Basic]

```
Overloads Public Function DeleteEntry() As Boolean
```

[C#]

```
public bool DeleteEntry();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The current phonebook entry name is specified by the **EntryName** property.

See Also

InternetDialer Class | SocketTools Namespace | InternetDialer.DeleteEntry Overload List

InternetDialer.DeleteEntry Method (String)

Delete a phonebook entry from the current phonebook.

[Visual Basic]

```
Overloads Public Function DeleteEntry( _
   ByVal entryName As String _
) As Boolean
```

[C#]

```
public bool DeleteEntry(
   string entryName
);
```

Parameters

entryName

A string which specifies the phonebook entry name to delete.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

See Also

InternetDialer Class | SocketTools Namespace | InternetDialer.DeleteEntry Overload List

InternetDialer.Disconnect Method

Terminate the connection with the dial-up networking service provider.

[Visual Basic]
```
Public Function Disconnect() As Boolean
```

[C#]
```
public bool Disconnect();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method may cause the current thread to block as the connection is terminated and the dial-up network device is being reset to its default state. Any active network connections using this dial-up networking connection will be terminated.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.Dispose Method

Releases all resources used by InternetDialer.

Overload List

Releases all resources used by InternetDialer.

public void Dispose();

Releases the unmanaged resources allocated by the InternetDialer class and optionally releases the managed resources.

protected virtual void Dispose(bool);

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.Dispose Method ()

Releases all resources used by InternetDialer.

[Visual Basic]

```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]

```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

InternetDialer Class | SocketTools Namespace | InternetDialer.Dispose Overload List

InternetDialer.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the InternetDialer class and optionally releases the managed resources.

[Visual Basic]
```
Overridable Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]
```
protected virtual void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **InternetDialer** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

InternetDialer Class | SocketTools Namespace | InternetDialer.Dispose Overload List

InternetDialer.EditEntry Method

Edit an existing phonebook entry in the current phonebook.

[Visual Basic]

```
Public Function EditEntry( _
   ByVal entryName As String _
) As Boolean
```

[C#]

```
public bool EditEntry(
   string entryName
);
```

Parameters

entryName
A string which specifies the name of the phonebook entry to be edited.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **EditEntry** method edits the specified entry from the local phonebook. This will cause a dialog box to be displayed from which the user can change the connection information. If you do not want to display a dialog, then use the **SaveEntry** method instead.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.Finalize Method

Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.

[Visual Basic]
```
Overrides Protected Sub Finalize()
```

[C#]
```
protected override void Finalize();
```

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.Initialize Method

Initialize an instance of the InternetDialer class.

Overload List

Initialize an instance of the InternetDialer class.

public bool Initialize();

Initialize an instance of the InternetDialer class.

public bool Initialize(string);

See Also

InternetDialer Class | SocketTools Namespace | Uninitialize Method

InternetDialer.Initialize Method ()

Initialize an instance of the InternetDialer class.

[Visual Basic]

```
Overloads Public Function Initialize() As Boolean
```

[C#]

```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the InternetDialer class, loading the networking library and allocating resources for the current thread. Typically it is not necessary to explicitly call this method because the instance of the class is initialized by the class constructor. However, if the **Uninitialize** method is called, the class must be re-initialized before any other methods are called.

See Also

InternetDialer Class | SocketTools Namespace | InternetDialer.Initialize Overload List | Uninitialize Method

InternetDialer.Initialize Method (String)

Initialize an instance of the InternetDialer class.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the InternetDialer class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the InternetDialer class.

Example

The following example shows how to use the Initialize method to initialize an instance of the class. This example assumes that the license key string has been defined in code.

```
SocketTools.InternetDialer xxxClient = new SocketTools.InternetDialer();

if (xxxClient.Initialize(strLicenseKey) == false)
{
    MessageBox.Show(xxxClient.LastErrorString, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    return;
}
```

```
Dim xxxClient As New SocketTools.InternetDialer

If xxxClient.Initialize(strLicenseKey) = False Then
    MsgBox(xxxClient.LastErrorString, vbIconExclamation)
    Exit Sub
End If
```

See Also

InternetDialer Class | SocketTools Namespace | InternetDialer.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

InternetDialer.LoadEntry Method

Reload the current phonebook entry from the phonebook.

Overload List

Reload the current phonebook entry from the phonebook.

public bool LoadEntry();

Load the specified entry from the current phonebook.

public bool LoadEntry(string);

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.LoadEntry Method ()

Reload the current phonebook entry from the phonebook.

[Visual Basic]
```
Overloads Public Function LoadEntry() As Boolean
```

[C#]
```
public bool LoadEntry();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The current entry from the phonebook will be reloaded and any changes made to the current entry will be abandoned.

The current phonebook entry name is specified by the **EntryName** property.

See Also

InternetDialer Class | SocketTools Namespace | InternetDialer.LoadEntry Overload List

InternetDialer.LoadEntry Method (String)

Load the specified entry from the current phonebook.

[Visual Basic]

```
Overloads Public Function LoadEntry( _
   ByVal entryName As String _
) As Boolean
```

[C#]

```
public bool LoadEntry(
   string entryName
);
```

Parameters

entryName

A string which specifies the name of a phonebook entry.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **LoadEntry** method loads the specified entry from the current phonebook and sets the class properties to match the configuration.

See Also

InternetDialer Class | SocketTools Namespace | InternetDialer.LoadEntry Overload List

InternetDialer.RenameEntry Method

Rename an existing phonebook entry.

[Visual Basic]

```
Public Function RenameEntry( _
   ByVal oldEntryName As String, _
   ByVal newEntryName As String _
) As Boolean
```

[C#]

```
public bool RenameEntry(
   string oldEntryName,
   string newEntryName
);
```

Parameters

oldEntryName
A string which specifies the phonebook entry which will be renamed.

newEntryName
A string which specifies the new name for the phonebook entry.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **RenameEntry** method renames the specified entry in the local phonebook. The new entry name must not already exist in the phonebook.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. If a socket has been allocated, it will be released and any active connections will be terminated. All properties will be reset to their default values.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.SaveEntry Method

Save the current phonebook entry settings.

Overload List

Save the current phonebook entry settings.

public bool SaveEntry();

Save the current settings to the specified phonebook entry in the current phonebook.

public bool SaveEntry(string);

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.SaveEntry Method ()

Save the current phonebook entry settings.

[Visual Basic]
```
Overloads Public Function SaveEntry() As Boolean
```

[C#]
```
public bool SaveEntry();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SaveEntry** method saves the current phonebook entry settings, based on the class property values. If the entry does not exist, it will be created. If the entry does exist, it will be overwritten. Note that unlike the **CreateEntry** method, this method does not display any user-interface dialogs.

The current phonebook entry name is specified by the **EntryName** property.

See Also

InternetDialer Class | SocketTools Namespace | InternetDialer.SaveEntry Overload List

InternetDialer.SaveEntry Method (String)

Save the current settings to the specified phonebook entry in the current phonebook.

[Visual Basic]

```
Overloads Public Function SaveEntry( _
   ByVal entryName As String _
) As Boolean
```

[C#]

```
public bool SaveEntry(
   string entryName
);
```

Parameters

entryName

A string which specifies the name of the phonebook entry.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SaveEntry** method saves the specified entry to the current phonebook, based on the class property values. If the entry does not exist, it will be created. If the entry does exist, it will be overwritten. Note that unlike the **CreateEntry** method, this method does not display any user-interface dialogs.

See Also

InternetDialer Class | SocketTools Namespace | InternetDialer.SaveEntry Overload List

InternetDialer.Uninitialize Method

Uninitialize the class library and release any resources allocated for the current thread.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method terminates any active connection, releases resources allocated for the current thread and unloads the networking library. After this method has been called, no further client operations may be performed until the class instance has been re-initialized.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the class is no longer needed.

See Also

InternetDialer Class | SocketTools Namespace | Initialize Method

InternetDialer Events

The events of the **InternetDialer** class are listed below. For a complete list of **InternetDialer** class members, see the InternetDialer Members topic.

Public Instance Events

⚡ OnCancel	Occurs when a blocking client operation is canceled.
⚡ OnConnect	Occurs when a connection is established with the service provider.
⚡ OnDisconnect	Occurs when the dial-up networking connection is terminated.
⚡ OnError	Occurs when an client operation fails.
⚡ OnStatus	Occurs when the when the connection state changes.
⚡ OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.OnCancel Event

Occurs when a blocking client operation is canceled.

[Visual Basic]

```
Public Event OnCancel As EventHandler
```

[C#]

```
public event EventHandler OnCancel;
```

Remarks

The **OnCancel** event is generated when a blocking client operation, such as sending or receiving data, is canceled with the **Cancel** method. To assist in determining which operation was canceled, check the value of the **Status** property.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.OnConnect Event

Occurs when a connection is established with the service provider.

[Visual Basic]

```
Public Event OnConnect As EventHandler
```

[C#]

```
public event EventHandler OnConnect;
```

Remarks

The **OnConnect** event occurs when a connection is made with a service provider as a result of a **Connect** method call. When the **Connect** method is called and the **Blocking** property is set to **false**, a dial-up networking connection is initiated, but the connection is not actually established until after this event occurs. Between the time connection process is started and this event fires, no operation may be performed by the client other than calling the **Disconnect** method.

This event is only generated if the client is in non-blocking mode.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.OnDisconnect Event

Occurs when the dial-up networking connection is terminated.

[Visual Basic]

```
Public Event OnDisconnect As EventHandler
```

[C#]

```
public event EventHandler OnDisconnect;
```

Remarks

The **OnDisconnect** event occurs when the dial-up networking connection has been terminated. This event is only generated if the client is in non-blocking mode.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.OnError Event

Occurs when an client operation fails.

[Visual Basic]

```
Public Event OnError As OnErrorEventHandler
```

[C#]

```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type InternetDialer.ErrorEventArgs containing data related to this event. The following **InternetDialer.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a client operation fails.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see InternetDialer.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.InternetDialer.ErrorEventArgs

[Visual Basic]
```
Public Class InternetDialer.ErrorEventArgs
    Inherits EventArgs
```

[C#]
```
public class InternetDialer.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetDialer (in SocketTools.InternetDialer.dll)

See Also

InternetDialer.ErrorEventArgs Members | SocketTools Namespace

InternetDialer.ErrorEventArgs Members

InternetDialer.ErrorEventArgs overview

Public Instance Constructors

InternetDialer.ErrorEventArgs Constructor	Initializes a new instance of the InternetDialer.ErrorEventArgs class.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetDialer.ErrorEventArgs Class | SocketTools Namespace

InternetDialer.ErrorEventArgs Constructor

Initializes a new instance of the InternetDialer.ErrorEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public InternetDialer.ErrorEventArgs();
```

See Also

InternetDialer.ErrorEventArgs Class | SocketTools Namespace

InternetDialer.ErrorEventArgs Properties

The properties of the **InternetDialer.ErrorEventArgs** class are listed below. For a complete list of **InternetDialer.ErrorEventArgs** class members, see the InternetDialer.ErrorEventArgs Members topic.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

InternetDialer.ErrorEventArgs Class | SocketTools Namespace

InternetDialer.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

InternetDialer.ErrorEventArgs Class | SocketTools Namespace | Error Property

InternetDialer.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public InternetDialer.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

InternetDialer.ErrorEventArgs Class | SocketTools Namespace | Description Property

InternetDialer.OnStatus Event

Occurs when the when the connection state changes.

[Visual Basic]

```
Public Event OnStatus As OnStatusEventHandler
```

[C#]

```
public event OnStatusEventHandler OnStatus;
```

Event Data

The event handler receives an argument of type InternetDialer.StatusEventArgs containing data related to this event. The following **InternetDialer.StatusEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a description of the current dial-up networking connection status.
Status	Gets a value which specifies the current status of the dial-up networking connection.

Remarks

The **OnStatus** event is generated when the status of the connection changes. Typically this occurs when a connection is being established with a dial-up networking server.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.StatusEventArgs Class

Provides data for the OnStatus event.

For a list of all members of this type, see InternetDialer.StatusEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.InternetDialer.StatusEventArgs

[Visual Basic]

```
Public Class InternetDialer.StatusEventArgs
    Inherits EventArgs
```

[C#]

```
public class InternetDialer.StatusEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

StatusEventArgs specifies the status code and a description of the status for the last status change that has occurred.

The OnStatus event occurs whenever the status of the dial-up networking connection changes.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetDialer (in SocketTools.InternetDialer.dll)

See Also

InternetDialer.StatusEventArgs Members | SocketTools Namespace

InternetDialer.StatusEventArgs Members

InternetDialer.StatusEventArgs overview

Public Instance Constructors

InternetDialer.StatusEventArgs Constructor	Initializes a new instance of the InternetDialer.StatusEventArgs class.

Public Instance Properties

Description	Gets a description of the current dial-up networking connection status.
Status	Gets a value which specifies the current status of the dial-up networking connection.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetDialer.StatusEventArgs Class | SocketTools Namespace

InternetDialer.StatusEventArgs Constructor

Initializes a new instance of the InternetDialer.StatusEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public InternetDialer.StatusEventArgs();
```

See Also

InternetDialer.StatusEventArgs Class | SocketTools Namespace

InternetDialer.StatusEventArgs Properties

The properties of the **InternetDialer.StatusEventArgs** class are listed below. For a complete list of **InternetDialer.StatusEventArgs** class members, see the InternetDialer.StatusEventArgs Members topic.

Public Instance Properties

Description	Gets a description of the current dial-up networking connection status.
Status	Gets a value which specifies the current status of the dial-up networking connection.

See Also

InternetDialer.StatusEventArgs Class | SocketTools Namespace

InternetDialer.StatusEventArgs.Description Property

Gets a description of the current dial-up networking connection status.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the connection status.

See Also

InternetDialer.StatusEventArgs Class | SocketTools Namespace | Error Property

InternetDialer.StatusEventArgs.Status Property

Gets a value which specifies the current status of the dial-up networking connection.

[Visual Basic]

```
Public ReadOnly Property Status As DialerStatus
```

[C#]

```
public InternetDialer.DialerStatus Status {get;}
```

Property Value

A DialerStatus enumeration value which specifies the current status.

See Also

InternetDialer.StatusEventArgs Class | SocketTools Namespace

InternetDialer.OnTimeout Event

Occurs when a blocking operation fails to complete before the timeout period elapses.

[Visual Basic]

```
Public Event OnTimeout As EventHandler
```

[C#]

```
public event EventHandler OnTimeout;
```

Remarks

The **OnTimeout** event occurs when a blocking operation, such as sending or receiving data on the client, fails to complete before the specified timeout period elapses. The timeout period for a blocking operation can be adjusted by setting the **Timeout** property.

This event is only generated if the client is in blocking mode.

See Also

InternetDialer Class | SocketTools Namespace

InternetDialer.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub InternetDialer.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void InternetDialer.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender
The source of the event.

e
An ErrorEventArgs that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetDialer (in SocketTools.InternetDialer.dll)

See Also

SocketTools Namespace

InternetDialer.OnStatusEventHandler Delegate

Represents the method that will handle the OnStatus event.

[Visual Basic]

```
Public Delegate Sub InternetDialer.OnStatusEventHandler( _
   ByVal sender As Object, _
   ByVal e As StatusEventArgs _
)
```

[C#]

```
public delegate void InternetDialer.OnStatusEventHandler(
      object sender,
      StatusEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A StatusEventArgs object that contains the event data.

Remarks

When you create an **OnStatusEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnStatusEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetDialer (in SocketTools.InternetDialer.dll)

See Also

SocketTools Namespace

InternetDialer.DialerStatus Enumeration

Specifies the status values that may be returned by the InternetDialer class.

[Visual Basic]
```
Public Enum InternetDialer.DialerStatus
```

[C#]
```
public enum InternetDialer.DialerStatus
```

Remarks

The InternetDialer class uses the **DialerStatus** enumeration to identify the current status of the client.

Members

Member Name	Description
statusUnused	No connection has been established.
statusOpenPort	The communications port is about to be opened.
statusPortOpened	The communications port has been opened.
statusConnectDevice	A device is about to be connected.
statusDeviceConnected	A device has been connected successfully.
statusAllDevicesConnected	All devices have been connected.
statusAuthenticate	Authenticating username and password.
statusAuthNotify	An authentication event has occurred.
statusAuthRetry	Requesting authentication with new credentials.
statusAuthCallback	The remote server has requested a callback number.
statusAuthChangePassword	The user has requested to change the password.
statusAuthProject	Registering computer on the network.
statusAuthLinkSpeed	The link speed calculation phase is starting.
statusAuthAck	An authentication request is being acknowledged.
statusReAuthenticate	Authenticating username and password.
statusAuthenticated	The user has been authenticated.
statusPrepareForCallback	The line is about to be disconnected in preparation for callback.
statusWaitForModemReset	The modem is resetting itself in preparation for callback.
statusWaitForCallback	Waiting for callback from remote server.
statusProjected	Protocol specific information has been negotiated.
statusStartAuthentication	User authentication is being initiated.
statusCallbackComplete	Callback completed and resuming authentication.

statusLogonNetwork	Logging on to the network.
statusSubEntryConnected	A subentry has been connected.
statusSubEntryDisconnected	A subentry has been disconnected.
statusInteractive	Initiating interactive login session.
statusRetryAuthentication	Retrying user authentication.
statusCallbackSetByCaller	Callback has been set by caller.
statusPasswordExpired	Password has expired.
statusConnected	Connected to remote server.
statusDisconnected	Disconnected from remote server.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetDialer (in SocketTools.InternetDialer.dll)

See Also

SocketTools Namespace

InternetDialer.ErrorCode Enumeration

Specifies the error codes returned by the InternetDialer class.

[Visual Basic]

```
Public Enum InternetDialer.ErrorCode
```

[C#]

```
public enum InternetDialer.ErrorCode
```

Remarks

The InternetDialer class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorDeviceNotFound	The specified device could not be found.
errorOperationTimeout	The specified operation has timed out.
errorPending	An operation is pending.
errorInvalidPortHandle	An invalid port handle was detected.
errorPortAlreadyOpen	The specified port is already open.
errorBufferTooSmall	The caller's buffer is too small.
errorWrongInfoSpecified	Incorrect information was specified.
errorCannotSetPortInfo	The port information cannot be set.
errorPortNotConnected	The specified port is not connected.
errorEventInvalid	An invalid event was detected.
errorDeviceDoesNotExist	A device was specified that does not exist.
errorDevicetypeDoesNotExist	A device type was specified that does not exist.
errorBufferInvalid	An invalid buffer was specified.
errorRouteNotAvailable	A route was specified that is not available.
errorRouteNotAllocated	A route was specified that is not allocated.
errorInvalidCompressionSpecified	An invalid compression was specified.
errorOutOfBuffers	There were insufficient buffers available.

errorPortNotFound	The specified port was not found.
errorAsyncRequestPending	An asynchronous request is pending.
errorAlreadyDisconnecting	The modem or other connecting device is already disconnecting.
errorPortNotOpen	The specified port is not open.
errorPortDisconnected	A connection to the remote computer could not be established.
errorNoEndpoints	No endpoints could be determined.
errorCannotOpenPhonebook	The system could not open the phone book file.
errorCannotLoadPhonebook	The system could not load the phone book file.
errorCannotFindPhonebookEntry	The system could not find the phone book entry for this connection.
errorCannotWritePhonebook	The system could not update the phone book file.
errorCorruptPhonebook	The system found invalid information in the phone book file.
errorCannotLoadString	A string could not be loaded.
errorKeyNotFound	A key could not be found.
errorDisconnection	The connection was terminated by the remote computer before it could be completed.
errorRemoteDisconnection	The connection was closed by the remote computer.
errorHardwareFailure	The modem or other connecting device was disconnected due to hardware failure.
errorUserDisconnection	The user disconnected the modem or other connecting device.
errorInvalidSize	An incorrect structure size was detected.
errorPortNotAvailable	The modem or other connecting device is already in use or is not configured properly.
errorCannotProjectClient	Your computer could not be registered on the remote network.
errorUnknown	There was an unknown error.
errorWrongDeviceAttached	The device attached to the port is not the one expected.
errorBadString	A string was detected that could not be converted.
errorRequestTimeout	The request has timed out.
errorCannotGetLana	No asynchronous net is available.
errorNetBIOSError	An error has occurred involving NetBIOS.
errorServerOutOfResources	The server cannot allocate NetBIOS resources needed to support the client.
errorNameExistsOnNet	One of your computer's NetBIOS names is already

	registered on the remote network.
errorServerGeneralNetFailure	A network adapter at the server failed.
errorMsgAliasNotAdded	You will not receive network message popups.
errorAuthInternal	There was an internal authentication error.
errorRestrictedLogonHours	The account is not permitted to log on at this time of day.
errorAcctDisabled	The account is disabled.
errorPasswdExpired	The password for this account has expired.
errorNoDialInPermission	The account does not have permission to dial in.
errorServerNotResponding	The remote access server is not responding.
errorFromDevice	The modem or other connecting device has reported an error.
errorUnrecognizedResponse	There was an unrecognized response from the modem or other connecting device.
errorMacroNotFound	A macro required by the modem or other connecting device was not found in the configuration file.
errorMacroNotDefined	A command or response in the configuration file refers to an undefined macro.
errorMessageMacroNotFound	The message macro was not found in the configuration file.
errorDefaultOffMacroNotFound	The configuration file contains an undefined macro.
errorFileCouldNotBeOpened	The configuration file could not be opened.
errorDevicenameTooLong	The device name in the configuration file is too long.
errorDevicenameNotFound	The configuration file refers to an unknown device name.
errorNoResponses	The configuration file contains no responses for the command.
errorNoCommandFound	The configuration file is missing a command.
errorWrongKeySpecified	There was an attempt to set a macro not listed in configuration file.
errorUnknownDeviceType	The configuration file refers to an unknown device type.
errorAllocatingMemory	The system has run out of memory.
errorPortNotConfigured	The modem or other connecting device is not properly configured.
errorDeviceNotReady	The modem or other connecting device is not functioning.

errorReadingIniFile	The system was unable to read the configuration file.
errorNoConnection	The connection was terminated.
errorBadUsageInIniFile	The usage parameter in the configuration file is invalid.
errorReadingSectionname	The system was unable to read the section name from the configuration file.
errorReadingDeviceType	The system was unable to read the device type from the configuration file.
errorReadingDeviceName	The system was unable to read the device name from the configuration file.
errorReadingUsage	The system was unable to read the usage from the configuration file.
errorReadingMaxConnectBps	The system was unable to read the maximum connection BPS rate from the configuration file.
errorReadingMaxCarrierBps	The system was unable to read the maximum carrier connection speed from the configuration file.
errorLineBusy	The phone line is busy.
errorVoiceAnswer	A person answered instead of a modem or other connecting device.
errorNoAnswer	The remote computer did not respond.
errorNoCarrier	The system could not detect the carrier.
errorNoDialtone	There was no dial tone.
errorInCommand	The modem or other connecting device reported a general error.
errorWritingSectionname	There was an error in writing the section name.
errorWritingDeviceType	There was an error in writing the device type.
errorWritingDeviceName	There was an error in writing the device name.
errorWritingMaxConnectBps	There was an error in writing the maximum connection speed..
errorWritingMaxCarrierBps	There was an error in writing the maximum carrier speed.
errorWritingUsage	There was an error in writing the usage.
errorWritingDefaultOff	There was an error in writing the default-off.
errorReadingDefaultOff	There was an error in reading the default-off.
errorEmptyIniFile	The configuration file is empty.
errorAuthenticationFailure	Access was denied because the username and/or password was invalid on the domain.
errorPortOrDevice	There was a hardware failure in the modem or other connecting device.

errorNotBinaryMacro	An internal error has occurred.
errorDcbNotFound	An internal error has occurred.
errorStateMachinesNotStarted	The state machines are not started.
errorStateMachinesAlreadyStarted	The state machines are already started.
errorPartialResponseLooping	The response looping did not complete.
errorUnknownResponseKey	A response keyname in the configuration file is not in the expected format.
errorRecvBufFull	The modem or other connecting device response caused a buffer overflow.
errorCmdTooLong	The expanded command in the configuration file is too long.
errorUnsupportedBps	The modem moved to a connection speed not supported by the COM driver.
errorUnexpectedResponse	Device response received when none expected.
errorInteractiveMode	The connection needs information from you, but the application does not allow user interaction.
errorBadCallbackNumber	The callback number is invalid.
errorInvalidAuthState	The authorization state is invalid.
errorWritingInitBps	An internal error has occurred.
errorX25Diagnostic	There was an error related to the X.25 protocol.
errorAcctExpired	The account has expired.
errorChangingPassword	There was an error changing the password on the domain.
errorOverrun	Serial overrun errors were detected while communicating with the modem.
errorRasmanCannotInitialize	A configuration error on this computer is preventing this connection.
errorBiplexPortNotAvailable	The two-way port is initializing, wait a few seconds and redial.
errorNoActiveIsdnLines	No active ISDN lines are available.
errorNoIsdnChannelsAvailable	No ISDN channels are available to make the call.
errorTooManyLineErrors	Too many errors occurred because of poor phone line quality.
errorIpConfiguration	The Remote Access Service IP configuration is unusable.
errorNoIpAddresses	No IP addresses are available in the static pool of Remote Access Service IP addresses.
errorPppTimeout	The connection was terminated because the remote computer did not respond in a timely manner.

errorPppRemoteTerminated	The connection was terminated by the remote computer.
errorPppNoProtocolsConfigured	A connection to the remote computer could not be established.
errorPppNoResponse	The remote computer did not respond.
errorPppInvalidPacket	Invalid data was received from the remote computer.
errorPhoneNumberTooLong	The phone number, including prefix and suffix, is too long.
errorIpxcpNoDialoutConfigured	The IPX protocol cannot dial out on the modem because this computer is not configured for dialing out.
errorIpxcpNoDialinConfigured	The IPX protocol cannot dial in on the modem because this computer is not configured for dialing in.
errorIpxcpDialoutAlreadyActive	The IPX protocol cannot be used for dialing out on more than one modem.
errorAccessingTcpcfgDll	Cannot access TCPCFG.DLL.
errorNoIpRasAdapter	The system cannot find an IP adapter.
errorSlipRequiresIp	SLIP cannot be used unless the IP protocol is installed.
errorProjectionNotComplete	Computer registration is not complete.
errorProtocolNotConfigured	The protocol is not configured.
errorPppNotConverging	Your computer and the remote computer could not agree on PPP control protocols.
errorPppCpRejected	A connection to the remote computer could not be completed.
errorPppLcpTerminated	The PPP link control protocol was terminated.
errorPppRequiredAddressRejected	The requested address was rejected by the server.
errorPppNcpTerminated	The remote computer terminated the control protocol.
errorPppLoopbackDetected	Loopback was detected.
errorPppNoAddressAssigned	The server did not assign an address.
errorCannotUseLogonCredentials	The authentication protocol required by the remote server cannot use the stored password.
errorTapiConfiguration	An invalid dialing rule was detected.
errorNoLocalEncryption	The local computer does not support the required data encryption type.
errorNoRemoteEncryption	The remote computer does not support the required data encryption type.
errorRemoteRequiresEncryption	The remote computer requires data encryption.

errorIpxcpNetNumberConflict	The system cannot use the IPX network number assigned by the remote computer.
errorInvalidSMM	An internal error has occurred.
errorSMMUninitialized	An internal error has occurred.
errorNoMacForPort	An internal error has occurred.
errorSmmTimeout	An internal error has occurred.
errorBadPhoneNumber	An invalid telephone number has been specified.
errorWrongModule	An internal error has occurred.
errorInvalidCallbackNumber	The callback number contains an invalid character.
errorScriptSyntax	A syntax error was encountered while processing a script.
errorHangupFailed	The connection could not be disconnected because it was created by the multi-protocol router.
errorBundleNotFound	The system could not find the multi-link bundle.
errorCannotDoCustomdial	The system cannot perform automated dial because this connection has a custom dialer specified.
errorDialAlreadyInProgress	This connection is already being dialed.
errorRasautoCannotInitialize	Remote Access Services could not be started automatically.
errorConnectionAlreadyShared	Internet Connection Sharing is already enabled on the connection.
errorSharingChangeFailed	An error occurred while the existing Internet Connection Sharing settings were being changed.
errorSharingRouterInstall	An error occurred while routing capabilities were being enabled.
errorShareConnectionFailed	An error occurred while Internet Connection Sharing was being enabled for the connection.
errorSharingPrivateInstall	An error occurred while the local network was being configured for sharing.
errorCannotShareConnection	Internet Connection Sharing cannot be enabled.
errorNoSmartCardReader	No smart card reader is installed.
errorSharingAddressExists	Internet Connection Sharing cannot be enabled.
errorNoCertificate	A certificate could not be found.
errorSharingMultipleAddresses	Internet Connection Sharing cannot be enabled.
errorFailedToEncrypt	The connection attempt failed because of failure to encrypt data.
errorBadAddressSpecified	The specified destination is not reachable.
errorConnectionReject	The remote computer rejected the connection

	attempt.
errorCongestion	The connection attempt failed because the network is busy.
errorIncompatible	The remote computer's network hardware is incompatible with the type of call requested.
errorNumberchanged	The connection attempt failed because the destination number has changed.
errorTempfailure	The connection attempt failed because of a temporary failure.
errorBlocked	The call was blocked by the remote computer.
errorDonotdisturb	The call could not be connected because the remote computer has invoked the Do Not Disturb feature.
errorOutoforder	The connection attempt failed because the modem on the remote computer is out of order.
errorUnableToAuthenticateServer	It was not possible to verify the identity of the server.
errorSmartCardRequired	To dial out using this connection you must use a smart card.
errorInvalidFunctionForEntry	An attempted function is not valid for this connection.
errorCertForEncryptionNotFound	The connection requires a certificate, and no valid certificate was found.
errorSharingRrasConflict	Network Address Translation must be removed before enabling Internet Connection Sharing.
errorSharingNoPrivateLan	Internet Connection Sharing cannot be enabled.
errorNoDiffUserAtLogon	You cannot dial using this connection at logon time.
errorNoRegCertAtLogon	You cannot dial using this connection at logon time.
errorOakleyNoCert	The L2TP connection attempt failed because there is no valid machine certificate on your computer for security authentication.
errorOakleyAuthFail	The L2TP connection attempt failed because the security layer could not authenticate the remote computer.
errorOakleyAttribFail	The L2TP connection attempt failed because the security layer could not negotiate compatible parameters with the remote computer.
errorOakleyGeneralProcessing	The L2TP connection attempt failed because the security layer encountered a processing error during initial negotiations with the remote computer.

errorOakleyNoPeerCert	The L2TP connection attempt failed because certificate validation on the remote computer failed.
errorOakleyNoPolicy	The L2TP connection attempt failed because security policy for the connection was not found.
errorOakleyTimedOut	The L2TP connection attempt failed because security negotiation timed out.
errorOakleyError	The L2TP connection attempt failed because an error occurred while negotiating security.
errorUnknownFramedProtocol	The Framed Protocol RADIUS attribute for this user is not PPP.
errorWrongTunnelType	The Tunnel Type RADIUS attribute for this user is not correct.
errorUnknownServiceType	The Service Type RADIUS attribute for this user is neither Framed nor Callback Framed.
errorConnectingDeviceNotFound	A connection to the remote computer could not be established because the modem was not found or was busy.
errorNoEaptlsCertificate	A certificate could not be found that can be used with this Extensible Authentication Protocol.
errorSharingHostAddressConflict	Internet Connection Sharing cannot be enabled.
errorAutomaticVpnFailed	Unable to establish the VPN connection.
errorValidatingServerCert	Unable to verify the digital certificate sent by the server.
errorReadSCard	The card supplied was not recognized, please check that the card is inserted correctly, and fits tightly
errorInvalidPEAPCookieConfig	The PEAP configuration stored in the session cookie does not match the current session configuration
errorInvalidPEAPCookieUser	The PEAP identity stored in the session cookie does not match the current identity
errorInvalidMSCHAPV2Config	You cannot dial using this connection at logon time, because it is configured to use logged on user's credentials
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorOperationNotSupported	The specified operation is not supported.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetDialer (in SocketTools.InternetDialer.dll)

See Also

SocketTools Namespace

InternetDialer.RasFramingProtocol Enumeration

Specifies the framing protocols supported by the InternetDialer class.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum InternetDialer.RasFramingProtocol
```

[C#]
```
[Flags]
public enum InternetDialer.RasFramingProtocol
```

Members

Member Name	Description	Value
framingProtocolPPP	Point-to-Point Protocol (PPP). This is the most common protocol used by Internet Service Providers (ISPs).	1
framingProtocolSLIP	Serial Line Internet Protocol (SLIP). This is a protocol commonly used when connecting to older UNIX systems.	2
framingProtocolRAS	A proprietary Microsoft protocol implemented in Windows for Workgroups 3.11 and Windows NT.	4

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetDialer (in SocketTools.InternetDialer.dll)

See Also

SocketTools Namespace

InternetDialer.RasNetworkProtocol Enumeration

Specifies the networking protocols supported by the InternetDialer class.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum InternetDialer.RasNetworkProtocol
```

```
[C#]
[Flags]
public enum InternetDialer.RasNetworkProtocol
```

Remarks

These values may be combined if multiple protocols should be negotiated when the connection is established. Note that unless there is a specific need for the application to use the NetBEUI or IPX protocols, it is recommended that only the TCP/IP protocol be specified.

Members

Member Name	Description	Value
networkProtocolNetBEUI	Negotiate the NetBEUI protocol.	1
networkProtocolIPX	Negotiate the IPX protocol.	2
networkProtocolIP	Negotiate the TCP/IP protocol.	4

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetDialer (in SocketTools.InternetDialer.dll)

See Also

SocketTools Namespace

InternetDialer.RasTerminalMode Enumeration

Specifies the interactive terminal modes supported by the InternetDialer class.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum InternetDialer.RasTerminalMode
```

[C#]

```
[Flags]
public enum InternetDialer.RasTerminalMode
```

Remarks

These values may be combined if multiple terminal modes should be used when the connection is established. If scripting has been enabled by setting the **ScriptFile** property, no terminal window should be displayed after the connection. This is because scripting has it's own terminal implementation.

Members

Member Name	Description	Value
terminalNone	No terminal window is displayed	0
terminalBeforeDial	Terminal window is displayed before dialing.	1
terminalAfterDial	Terminal window is displayed after dialing. Do not use if scripting has been enabled.	2

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetDialer (in SocketTools.InternetDialer.dll)

See Also

SocketTools Namespace

InternetDialer.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see InternetDialer.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.InternetDialer.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class InternetDialer.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class InternetDialer.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the InternetDialer class.

Example

```
<Assembly: SocketTools.InternetDialer.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.InternetDialer.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetDialer (in SocketTools.InternetDialer.dll)

See Also

InternetDialer.RuntimeLicenseAttribute Members | SocketTools Namespace

InternetDialer.RuntimeLicenseAttribute Members

InternetDialer.RuntimeLicenseAttribute overview

Public Instance Constructors

InternetDialer.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetDialer.RuntimeLicenseAttribute Class | SocketTools Namespace

InternetDialer.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public InternetDialer.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the InternetDialer class.

See Also

InternetDialer.RuntimeLicenseAttribute Class | SocketTools Namespace

InternetDialer.RuntimeLicenseAttribute Properties

The properties of the **InternetDialer.RuntimeLicenseAttribute** class are listed below. For a complete list of **InternetDialer.RuntimeLicenseAttribute** class members, see the InternetDialer.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

InternetDialer.RuntimeLicenseAttribute Class | SocketTools Namespace

InternetDialer.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

InternetDialer.RuntimeLicenseAttribute Class | SocketTools Namespace

InternetDialerException Class

The exception that is thrown when a client error occurs.

For a list of all members of this type, see InternetDialerException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.InternetDialerException

[Visual Basic]

```
Public Class InternetDialerException
    Inherits ApplicationException
```

[C#]

```
public class InternetDialerException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A InternetDialerException is thrown by the InternetDialer class when an error occurs.

The default constructor for the InternetDialerException class sets the **ErrorCode** property to the last client error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetDialer (in SocketTools.InternetDialer.dll)

See Also

InternetDialerException Members | SocketTools Namespace

InternetDialerException Members

InternetDialerException overview

Public Instance Constructors

InternetDialerException	Overloaded. Initializes a new instance of the InternetDialerException class.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetDialerException Class | SocketTools Namespace

InternetDialerException Constructor

Initializes a new instance of the InternetDialerException class with the last network error code.

Overload List

Initializes a new instance of the InternetDialerException class with the last network error code.

public InternetDialerException();

Initializes a new instance of the InternetDialerException class with a specified error number.

public InternetDialerException(int);

Initializes a new instance of the InternetDialerException class with a specified error message.

public InternetDialerException(string);

Initializes a new instance of the InternetDialerException class with a specified error message and a reference to the inner exception that is the cause of this exception.

public InternetDialerException(string,Exception);

See Also

InternetDialerException Class | SocketTools Namespace

InternetDialerException Constructor ()

Initializes a new instance of the InternetDialerException class with the last network error code.

[Visual Basic]
```
Overloads Public Sub New()
```

[C#]
```
public InternetDialerException();
```

Remarks

The ctor constructor sets the **ErrorCode** property to the last client error that occurred. For more information about the errors that may occur, refer to the InternetDialer.ErrorCode enumeration.

See Also

InternetDialerException Class | SocketTools Namespace | InternetDialerException Constructor Overload List

InternetDialerException Constructor (String)

Initializes a new instance of the InternetDialerException class with a specified error message.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]

```
public InternetDialerException(
   string message
);
```

Parameters

message

The error message that explains the reason for the exception.

Remarks

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

InternetDialerException Class | SocketTools Namespace | InternetDialerException Constructor Overload List

InternetDialerException Constructor (String, Exception)

Initializes a new instance of the InternetDialerException class with a specified error message and a reference to the inner exception that is the cause of this exception.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal innerException As Exception _
)
```

[C#]

```
public InternetDialerException(
   string message,
   Exception innerException
);
```

Parameters

message

The error message that explains the reason for the exception.

innerException

The exception that is the cause of the current exception. If the *innerException* parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

InternetDialerException Class | SocketTools Namespace | InternetDialerException Constructor Overload List

InternetDialerException Constructor (Int32)

Initializes a new instance of the InternetDialerException class with a specified error number.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

[C#]

```
public InternetDialerException(
   int code
);
```

Parameters

code

An integer value which specifies an error code.

Remarks

This constructor sets the **ErrorCode** property to the specified error code. For more information about the errors that may occur, refer to the InternetDialer.ErrorCode enumeration.

See Also

InternetDialerException Class | SocketTools Namespace | InternetDialerException Constructor Overload List

InternetDialerException Properties

The properties of the **InternetDialerException** class are listed below. For a complete list of **InternetDialerException** class members, see the InternetDialerException Members topic.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

InternetDialerException Class | SocketTools Namespace

InternetDialerException.ErrorCode Property

Gets a value which specifies the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property ErrorCode As ErrorCode
```

[C#]

```
public InternetDialer.ErrorCode ErrorCode {get;}
```

Property Value

Returns a InternetDialer.ErrorCode enumeration value which specifies the error code.

Remarks

The **ErrorCode** property returns the error code that specifies the cause of the exception.

The default constructor for the InternetDialerException class sets the error code to the last network error that occurred. For more information about the errors that may occur, refer to the InternetDialer.ErrorCode enumeration.

See Also

InternetDialerException Class | SocketTools Namespace

InternetDialerException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]
```
Overrides Public ReadOnly Property Message As String
```

[C#]
```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

InternetDialerException Class | SocketTools Namespace

InternetDialerException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception. This value is the same as the values returned by the Windows Sockets API. For more information about socket error codes, see the Windows Socket Version 2 API error code documentation in MSDN.

See Also

InternetDialerException Class | SocketTools Namespace

InternetDialerException Methods

The methods of the **InternetDialerException** class are listed below. For a complete list of **InternetDialerException** class members, see the InternetDialerException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetDialerException Class | SocketTools Namespace

InternetDialerException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]
```
Overrides Public Function ToString() As String
```

[C#]
```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

InternetDialerException Class | SocketTools Namespace

InternetMail Class

The InternetMail class enables a developer to create, send and retrieve email messages. The class implements the Simple Mail Transfer Protocol (SMTP) for sending messages, the Post Office Protocol (POP3) for retrieving messages from a mail server and the Multipurpose Internet Mail Extensions (MIME) standard for composing messages.

For a list of all members of this type, see InternetMail Members.

System.Object
  SocketTools.InternetMail

[Visual Basic]

```
Public Class InternetMail
    Implements IDisposable
```

[C#]

```
public class InternetMail : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The InternetMail class provides a simplified interface for composing, sending and retrieving email messages. The class was designed for ease-of-use and flexibility, without the inherent learning curve and additional coding required when using multiple components. In many cases, email functionality is simply one feature in an already complex project. Instead of setting dozens of properties and writing many lines of code to connect the output of one control to the input of another, the InternetMail class requires just two method calls to compose and deliver a message. The simple elegance of the component's interface translates directly into fewer lines of source code to write, debug and maintain. In turn, this allows the developer to focus his efforts on the core application without sacrificing features that add value to the end-user.

The class offers a comprehensive interface, providing the developer with everything that he needs to incorporate email functionality in an application. Many of the class' properties control the contents of a message, such as the list of recipients, the subject of the message and the message body. Methods are used to compose new messages, retrieve messages from a mail server and deliver messages to one or more recipients. Messages can also be managed on the mail server, or downloaded to the local system and stored in a file or a database record. The developer has complete access to all of the headers in the message, and can create custom application-specific header fields if needed. Event notifications enable the developer to provide the user with feedback, such as the progress of sending or retrieving a message. Advanced features such as delivery status notification, support for relay servers and secure encrypted connections are easily implemented by simply setting a few properties.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

InternetMail Members | SocketTools Namespace

InternetMail Members

InternetMail overview

Public Static (Shared) Fields

mailPortImap4	A constant value which specifies the default port number for a connection to a mail server.
mailPortImap4s	A constant value which specifies the default port number for a secure connection to a mail server.
mailPortPop3	A constant value which specifies the default port number for a connection to a mail server.
mailPortPop3s	A constant value which specifies the default port number for a secure connection to a mail server.
mailPortSmtp	A constant value which specifies the default port number for submitting messages to a mail server.
mailTimeout	A constant value which specifies the default timeout period.

Public Instance Constructors

InternetMail Constructor	Initializes a new instance of the InternetMail class.

Public Instance Fields

Mailbox	Gets the names of the available mailboxes for the current user.
NameServer	Change or return the Internet address for a nameserver
Recipient	Gets the recipients specified in the current message.

Public Instance Properties

AllHeaders	Gets a value which returns a list of all message recipients.
AllRecipients	Gets a value which returns a list of all message recipients.
Attachment	Gets and sets the name of the current file attachment.
Bcc	Gets and sets the blind carbon-copy header field value.
Cc	Gets and sets the carbon-copy header field value.
CertificateExpires	Get a value that specifies the date that the security certificate expires.
CertificateIssued	Get a value that specifies the date that the security certificate was issued.

CertificateIssuer	Get a value that provides information about the organization that issued the certificate.
CertificateName	Gets and sets a value that specifies the name of the client certificate.
CertificatePassword	Gets and sets the password associated with the client certificate.
CertificateStatus	Gets a value which indicates the status of the security certificate returned by the remote host.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateSubject	Gets a value that provides information about the organization that the server certificate was issued to.
CertificateUser	Gets and sets the user that owns the client certificate.
CipherStrength	Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.
ContentId	Gets the content identifier for the current message part.
ContentLength	Gets the size of the data stored in the current message part.
ContentType	Gets and sets the content type of the selected message part.
Date	Gets and sets the date for the current message.
Delivered	Gets the number of recipients the message has been delivered to.
Domain	Gets and sets the local domain name.
Encoding	Gets and sets the content encoding method used for the current message part.
From	Gets and sets the address of the user who sent the message.
HashStrength	Gets a value which specifies the length of the message digest that was selected for a secure connection.
IsConnected	Gets a value which indicates if a connection to the mail server has been established.
IsIdle	Gets a value which indicates if the client is idle and the current mailbox is being monitored for status changes.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
LastError	Gets and sets a value which specifies the last error

	that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LastMessage	Gets the number of the last message available in the current mailbox.
LocalAddress	Gets and sets the local Internet address that the client will be bound to.
Localize	Gets and sets a value which specifies if date and time values should be localized.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
Mailboxes	Gets the number of mailboxes available on the server.
MailboxFlags	Gets one or more flags which identify characteristics of the current mailbox.
MailboxMask	Gets and sets the current mailbox wildcard mask.
MailboxName	Gets and sets the name of the current mailbox.
MailboxPath	Gets and sets the current mailbox reference path.
MailboxSize	Gets the size of the current mailbox.
MailboxUID	Gets the unique identifier for the current mailbox.
Mailer	Gets and sets the name of the mailer application.
Message	Gets and sets the current message headers and body.
MessageCount	Gets the number of messages available in the current mailbox.
MessageFlags	Gets and sets one or more flags for the current message.
MessageID	Gets the current message identifier.
MessageIndex	Gets and sets the current message number.
MessagePart	Gets and sets the current section index in a multipart message.
MessageParts	Gets the number of sections in a multipart message.
MessageSize	Gets the size of the current message in bytes.
MessageText	Gets and sets the text body of the current message part.
MessageUID	Gets the UID for the current message.
MimeVersion	Gets and sets the MIME version number for the current message.

NewMessages	Gets the number of new messages available in the current mailbox.
Options	Gets and sets a value which specifies one or more client options.
Organization	Gets and sets the name of the organization that originated the message.
Password	Gets and sets the password used to authenticate the client.
Priority	Gets and sets the current message priority.
RecentMessages	Gets the number of messages which have recently arrived in the mailbox.
Recipients	Gets the number of recipients specified in the current message.
RelayPort	Gets and sets a value which specifies the relay server port number.
RelayServer	Gets and sets a value which specifies the relay server name or address.
ReplyTo	Gets and sets the address of the user who should receive replies to this message.
ReturnReceipt	Gets and sets the address of the person who should receive a message indicating that the message has been read.
Secure	Gets and sets a value which specifies if a secure connection is established.
SecureCipher	Gets a value that specifies the encryption algorithm used for a secure connection.
SecureHash	Gets a value that specifies the message digest algorithm used for a secure connection.
SecureKeyExchange	Gets a value that specifies the key exchange algorithm used for a secure connection.
SecureProtocol	Gets and sets a value which specifies the protocol used for a secure connection.
Sender	Gets and sets the address of the user who originated the message.
ServerName	Gets and sets a value which specifies the host name or address of the mail server.
ServerPort	Gets and sets a value which specifies the remote port number.
ServerType	Gets and sets a value which specifies the type of mail server the client is connecting to.
Subject	Gets and sets the subject of the current message.
Subscribed	Gets a value that specifies if the user has

	subscribed to the currently selected mailbox.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
TimeZone	Gets and sets the current timezone offset in seconds.
To	Gets and sets the address of the message recipient.
Trace	Gets and sets a value which indicates if network function tracing is enabled.
TraceFile	Gets and sets a value which specifies the name of the client function tracing logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
UserName	Gets and sets the username used to authenticate the client session.
Version	Gets a value which returns the current version of the InternetMail class library.

Public Instance Methods

AppendMessage	Append text to the body of the current message part.
AttachData	Overloaded. Attach the contents of a byte array to the current message.
AttachFile	Overloaded. Attach the specified file to the current message.
AttachImage	Overloaded. Attach an inline image to the current message.
AttachThread	Obsolete. Attach an instance of the class to the current thread.
Cancel	Cancel the current blocking client operation.
ChangePassword	Change the mailbox password for the current user.
CheckMailbox	Create a checkpoint for the currently selected mailbox.
ClearMessage	Clear the header and body of the current message.
ComposeMessage	Overloaded. Compose a new mail message.
Connect	Overloaded. Establish a connection with a mail server.
CopyMessage	Copy a message from the current mailbox to

	another mailbox.
CreateMailbox	Creates a new mailbox on the server.
CreateMessage	Overloaded. Create a new message.
CreatePart	Overloaded. Create a new message part in a multipart message.
DeleteHeader	Overloaded. Delete a header field from the specified message part.
DeleteMailbox	Overloaded. Deletes a mailbox from the server.
DeleteMessage	Flags a message for deletion from the current mailbox.
DeletePart	Delete the specified message part from the current message.
Disconnect	Terminate the connection with the remote server.
Dispose	Overloaded. Releases all resources used by InternetMail.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
ExportMessage	Overloaded. Export the current message to a file on the local system.
ExtractAllFiles	Overloaded. Extract all file attachments from the current message.
ExtractFile	Overloaded. Extract the contents of a file attachment and store it on the local system.
FindAttachment	Search for a specific file attachment in the current message.
GetFirstHeader	Return the first header in the current message part.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetHeader	Overloaded. Return the value of a header field in the specified message part.
GetHeaders	Overloaded. Retrieves the headers for the specified message from the server.
GetMessage	Overloaded. Retrieve a message from the server and return the contents in a byte array.
GetNextHeader	Return the next header in the current message part.
GetType (inherited from Object)	Gets the Type of the current instance.
Idle	Overloaded. Enables mailbox status monitoring for the client session.

ImportMessage	Replace the current message with the contents of a file.
Initialize	Overloaded. Initialize an instance of the InternetMail class.
ParseAddress	Overloaded. Parse an Internet email address.
ParseMessage	Parse the specified string, adding the contents to the current message.
RenameMailbox	Change the name of a mailbox.
ReselectMailbox	Reselects the current mailbox.
Reset	Reset the internal state of the object, resetting all properties to their default values.
SearchMailbox	Overloaded. Search the current mailbox for messages that match the specified criteria and character set.
SelectMailbox	Selects the specified mailbox for read-write access.
SendMessage	Overloaded. Submit the specified message to a mail server for delivery.
SetHeader	Overloaded. Set the value for a header in the specified message part.
StoreMessage	Overloaded. Retrieve a message from the current mailbox and store it in a file on the local system.
SubscribeMailbox	Overloaded. Subscribes the user to the specified mailbox.
ToString (inherited from Object)	Returns a String that represents the current Object.
UndeleteMessage	Removes the deletion flag for the specified message.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
UnselectMailbox	Overloaded. Unselects the current mailbox.
UnsubscribeMailbox	Overloaded. Unsubscribes the user from the specified mailbox.

Public Instance Events

OnCancel	Occurs when a blocking client operation is canceled.
OnDelivered	Occurs when a message has been submitted for delivery.
OnError	Occurs when an client operation fails.
OnProgress	Occurs as a data stream is being read or written to the client.
OnRecipient	Occurs when a message is about to be submitted for delivery.

OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
OnUpdate	Occurs when the server sends a mailbox update notification to the client.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the InternetMail class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the current message.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetMail Class | SocketTools Namespace

InternetMail Constructor

Initializes a new instance of the InternetMail class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public InternetMail();
```

See Also

InternetMail Class | SocketTools Namespace

InternetMail Fields

The fields of the **InternetMail** class are listed below. For a complete list of **InternetMail** class members, see the InternetMail Members topic.

Public Static (Shared) Fields

mailPortImap4	A constant value which specifies the default port number for a connection to a mail server.
mailPortImap4s	A constant value which specifies the default port number for a secure connection to a mail server.
mailPortPop3	A constant value which specifies the default port number for a connection to a mail server.
mailPortPop3s	A constant value which specifies the default port number for a secure connection to a mail server.
mailPortSmtp	A constant value which specifies the default port number for submitting messages to a mail server.
mailTimeout	A constant value which specifies the default timeout period.

Public Instance Fields

Mailbox	Gets the names of the available mailboxes for the current user.
NameServer	Change or return the Internet address for a nameserver
Recipient	Gets the recipients specified in the current message.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.mailPortImap4 Field

A constant value which specifies the default port number for a connection to a mail server.

[Visual Basic]

```
Public Const mailPortImap4 As Integer = 143
```

[C#]

```
public const int mailPortImap4 = 143;
```

Remarks

The default port number for the Internet Message Access Protocol is 143.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.mailPortImap4s Field

A constant value which specifies the default port number for a secure connection to a mail server.

[Visual Basic]

```
Public Const mailPortImap4s As Integer = 993
```

[C#]

```
public const int mailPortImap4s = 993;
```

Remarks

The default port number for a secure connection using IMAP4 over SSL/TLS is 993.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.mailPortPop3 Field

A constant value which specifies the default port number for a connection to a mail server.

[Visual Basic]

```
Public Const mailPortPop3 As Integer = 110
```

[C#]

```
public const int mailPortPop3 = 110;
```

Remarks

The default port number for the Post Office Protocol is 110.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.mailPortPop3s Field

A constant value which specifies the default port number for a secure connection to a mail server.

[Visual Basic]

```
Public Const mailPortPop3s As Integer = 995
```

[C#]

```
public const int mailPortPop3s = 995;
```

Remarks

The default port number for a secure connection using POP3 over SSL/TLS is 995.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.mailPortSmtp Field

A constant value which specifies the default port number for submitting messages to a mail server.

[Visual Basic]

```
Public Const mailPortSmtp As Integer = 25
```

[C#]

```
public const int mailPortSmtp = 25;
```

Remarks

The default port number for the Simple Mail Transfer Protocol is 25.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.mailTimeout Field

A constant value which specifies the default timeout period.

[Visual Basic]

```
Public Const mailTimeout As Integer = 20
```

[C#]

```
public const int mailTimeout = 20;
```

Remarks

The default timeout period is 20 seconds for each blocking network operation. An error will occur if the operation does not complete within the specified time period.

See Also

InternetMail Class | SocketTools Namespace

InternetMail Properties

The properties of the **InternetMail** class are listed below. For a complete list of **InternetMail** class members, see the InternetMail Members topic.

Public Instance Properties

AllHeaders	Gets a value which returns a list of all message recipients.
AllRecipients	Gets a value which returns a list of all message recipients.
Attachment	Gets and sets the name of the current file attachment.
Bcc	Gets and sets the blind carbon-copy header field value.
Cc	Gets and sets the carbon-copy header field value.
CertificateExpires	Get a value that specifies the date that the security certificate expires.
CertificateIssued	Get a value that specifies the date that the security certificate was issued.
CertificateIssuer	Get a value that provides information about the organization that issued the certificate.
CertificateName	Gets and sets a value that specifies the name of the client certificate.
CertificatePassword	Gets and sets the password associated with the client certificate.
CertificateStatus	Gets a value which indicates the status of the security certificate returned by the remote host.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateSubject	Gets a value that provides information about the organization that the server certificate was issued to.
CertificateUser	Gets and sets the user that owns the client certificate.
CipherStrength	Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.
ContentId	Gets the content identifier for the current message part.
ContentLength	Gets the size of the data stored in the current message part.
ContentType	Gets and sets the content type of the selected message part.

Date	Gets and sets the date for the current message.
Delivered	Gets the number of recipients the message has been delivered to.
Domain	Gets and sets the local domain name.
Encoding	Gets and sets the content encoding method used for the current message part.
From	Gets and sets the address of the user who sent the message.
HashStrength	Gets a value which specifies the length of the message digest that was selected for a secure connection.
IsConnected	Gets a value which indicates if a connection to the mail server has been established.
IsIdle	Gets a value which indicates if the client is idle and the current mailbox is being monitored for status changes.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LastMessage	Gets the number of the last message available in the current mailbox.
LocalAddress	Gets and sets the local Internet address that the client will be bound to.
Localize	Gets and sets a value which specifies if date and time values should be localized.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
Mailboxes	Gets the number of mailboxes available on the server.
MailboxFlags	Gets one or more flags which identify characteristics of the current mailbox.
MailboxMask	Gets and sets the current mailbox wildcard mask.
MailboxName	Gets and sets the name of the current mailbox.
MailboxPath	Gets and sets the current mailbox reference path.
MailboxSize	Gets the size of the current mailbox.
MailboxUID	Gets the unique identifier for the current mailbox.
Mailer	Gets and sets the name of the mailer application.

Message	Gets and sets the current message headers and body.
MessageCount	Gets the number of messages available in the current mailbox.
MessageFlags	Gets and sets one or more flags for the current message.
MessageID	Gets the current message identifier.
MessageIndex	Gets and sets the current message number.
MessagePart	Gets and sets the current section index in a multipart message.
MessageParts	Gets the number of sections in a multipart message.
MessageSize	Gets the size of the current message in bytes.
MessageText	Gets and sets the text body of the current message part.
MessageUID	Gets the UID for the current message.
MimeVersion	Gets and sets the MIME version number for the current message.
NewMessages	Gets the number of new messages available in the current mailbox.
Options	Gets and sets a value which specifies one or more client options.
Organization	Gets and sets the name of the organization that originated the message.
Password	Gets and sets the password used to authenticate the client.
Priority	Gets and sets the current message priority.
RecentMessages	Gets the number of messages which have recently arrived in the mailbox.
Recipients	Gets the number of recipients specified in the current message.
RelayPort	Gets and sets a value which specifies the relay server port number.
RelayServer	Gets and sets a value which specifies the relay server name or address.
ReplyTo	Gets and sets the address of the user who should receive replies to this message.
ReturnReceipt	Gets and sets the address of the person who should receive a message indicating that the message has been read.
Secure	Gets and sets a value which specifies if a secure connection is established.

SecureCipher	Gets a value that specifies the encryption algorithm used for a secure connection.
SecureHash	Gets a value that specifies the message digest algorithm used for a secure connection.
SecureKeyExchange	Gets a value that specifies the key exchange algorithm used for a secure connection.
SecureProtocol	Gets and sets a value which specifies the protocol used for a secure connection.
Sender	Gets and sets the address of the user who originated the message.
ServerName	Gets and sets a value which specifies the host name or address of the mail server.
ServerPort	Gets and sets a value which specifies the remote port number.
ServerType	Gets and sets a value which specifies the type of mail server the client is connecting to.
Subject	Gets and sets the subject of the current message.
Subscribed	Gets a value that specifies if the user has subscribed to the currently selected mailbox.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
TimeZone	Gets and sets the current timezone offset in seconds.
To	Gets and sets the address of the message recipient.
Trace	Gets and sets a value which indicates if network function tracing is enabled.
TraceFile	Gets and sets a value which specifies the name of the client function tracing logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
UserName	Gets and sets the username used to authenticate the client session.
Version	Gets a value which returns the current version of the InternetMail class library.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.AllHeaders Property

Gets a value which returns a list of all message recipients.

[Visual Basic]

```
Public ReadOnly Property AllHeaders As String
```

[C#]

```
public string AllHeaders {get;}
```

Property Value

A string which contains the complete RFC822 headers for the message.

Remarks

The **AllHeaders** property will return all of the RFC 822 header values in a string. This includes the message headers that are most commonly referred to, such as the To, From and Subject headers. Each header and its value are separated by a colon, and terminated with a carriage return and linefeed (CRLF) pair.

The headers and their values returned by this property will not be identical to the header block in the original message. If a header value is split across multiple lines, the text returned by this property will be folded, with the complete header value on a single line of text and removing any extraneous whitespace. If the header value has been encoded by the mail client, this property will return the decoded value, not the original encoded value.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.AllRecipients Property

Gets a value which returns a list of all message recipients.

[Visual Basic]
```
Public ReadOnly Property AllRecipients As String
```

[C#]
```
public string AllRecipients {get;}
```

Property Value

A string which contains a comma-separated list of all message recipients.

Remarks

The **AllRecipients** property returns a string value that contains a comma-separated list of all message recipients. To individually enumerate through each of the recipient addresses, you can use the **Recipient** property array and **Recipients** property.

Note that this property value will include those addresses specified by the **Bcc** property, even though they are not included in the message header.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Attachment Property

Gets and sets the name of the current file attachment.

[Visual Basic]

```
Public ReadOnly Property Attachment As String
```

[C#]

```
public string Attachment {get;}
```

Property Value

A string which specifies the name of an attached file.

Remarks

The **Attachment** property specifies the name of the file attachment in a multipart message. When a new part is selected that contains an attached file, the **Attachment** property is updated to reflect the attached file's name.

This property is used by the attach and extract actions to specify the local file name that will be used. Changing its value does not change the attached file name in the multipart message itself.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Bcc Property

Gets and sets the blind carbon-copy header field value.

[Visual Basic]

```
Public Property Bcc As String
```

[C#]

```
public string Bcc {get; set;}
```

Property Value

A string which specifies one or more blind carbon-copy recipients.

Remarks

The **Bcc** property returns the list of addresses that are to receive blind carbon copies of the message. Setting the property creates or modifies the Bcc header field. Multiple addresses can be specified by separating them with commas.

A blind carbon copy is when a copy of a message is delivered to a recipient, but that recipient is not listed in the message headers. Because the other recipients of that same message will not see the address in the headers, they will not know it was delivered to that person. As a result, the Bcc header field is not normally exported when the **ExportMessage** method is called, or when the contents of the message are referenced using the **Message** property. To include the Bcc header in the message, use the **exportAllHeaders** option. Of course, if this option is specified, the addresses in the Bcc list will no longer be blind to the other recipients.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.BearerToken Property

Gets and sets the bearer token used with OAuth 2.0 authentication.

[Visual Basic]

```
Public Property BearerToken As String
```

[C#]

```
public string BearerToken {get; set;}
```

Property Value

Returns a string which contains the bearer token. Assigning a value to this property sets the curent authentication type to use OAuth 2.0 authentication and updates the bearer token.

Remarks

Assigning a value to the **BearerToken** property will automatically change the current authentication method to use OAuth 2.0 and will clear the current **Password** property value.

You should only use an OAuth 2.0 authentication method if you understand the process of how to request the access token. Obtaining a bearer token requires registering your application with the mail service provider (e.g.: Microsoft or Google), getting a unique client ID associated with your application and then requesting the bearer token using the appropriate scope for the service. Obtaining the initial token will typically involve interactive confirmation on the part of the user, requiring they grant permission to your application to access their mail account.

Your application should not store the bearer token for later use. They usually have a relatively short lifespan, typically about an hour, and are designed to be used with the current client session. You should specify offline access as part of the OAuth 2.0 scope, and store the refresh token provided by the service. The refresh token has a much onger validity period and can be used to obtain a new access token when needed.

If the current authentication method does not use OAuth 2.0, this property will return an empty string and you should use the **Password** property to obtain the current user password.

See Also

InternetMail Class | SocketTools Namespace | Password Property | UserName Property | Connect Method

InternetMail.Cc Property

Gets and sets the carbon-copy header field value.

[Visual Basic]

```
Public Property Cc As String
```

[C#]

```
public string Cc {get; set;}
```

Property Value

A string which specifies one or more carbon-copy recipients.

Remarks

The **Cc** property returns the list of addresses that were delivered carbon copies of the message. Setting the property creates or modifies the Cc header field. Multiple addresses can be specified by separating them with commas.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.CertificateExpires Property

Get a value that specifies the date that the security certificate expires.

[Visual Basic]

```
Public ReadOnly Property CertificateExpires As String
```

[C#]

```
public string CertificateExpires {get;}
```

Property Value

A string which specifies a date using the local date and time format.

Remarks

The **CertificateExpires** property returns a string that specifies the date and time that the security certificate expires. This property will return an empty string if a secure connection has not been established with the remote host.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.CertificateIssued Property

Get a value that specifies the date that the security certificate was issued.

[Visual Basic]

```
Public ReadOnly Property CertificateIssued As String
```

[C#]

```
public string CertificateIssued {get;}
```

Property Value

A string which specifies a date using the local date and time format.

Remarks

The **CertificateIssued** property returns a string that specifies the date and time that the security certificate was issued. This property will return an empty string if a secure connection has not been established with the remote host.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.CertificateIssuer Property

Get a value that provides information about the organization that issued the certificate.

[Visual Basic]

```
Public ReadOnly Property CertificateIssuer As String
```

[C#]

```
public string CertificateIssuer {get;}
```

Property Value

A string that contains a comma separated list of name value pairs.

Remarks

The **CertificateIssuer** property returns a string that contains information about the organization that issued the server certificate. The string value is a comma separated list of tagged name and value pairs. In the nomenclature of the X.500 standard, each of these pairs are called a relative distinguished name (RDN), and when concatenated together, forms the issuer's distinguished name (DN). For example:

```
C=US, O="RSA Data Security, Inc.", OU=Secure Server Certification Authority
```

To obtain a specific value, such as the name of the issuer or the issuer's country, the application must parse the string returned by this property. Some of the common tokens used in the distinguished name are:

Token	Description
C	The ISO standard two character country code.
S	The name of the state or province.
L	The name of the city or locality.
O	The name of the company or organization.
OU	The name of the department or organizational unit
CN	The common name; with X.509 certificates, this is the domain name of the site the certificate was issued for.

This property will return an empty string if a secure connection has not been established with the remote host.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.CertificateName Property

Gets and sets a value that specifies the name of the client certificate.

[Visual Basic]

```
Public Property CertificateName As String
```

[C#]

```
public string CertificateName {get; set;}
```

Property Value

A string which specifies the certificate name.

Remarks

The **CertificateName** property is used to specify the name of a client certificate to use when establishing a secure connection. It is only required that you set this property value if the server requires a client certificate for authentication. If this property is not set, a client certificate will not be provided to the server. If a certificate name is specified, the certificate must have a private key associated with it, otherwise the connection attempt will fail because the control will be unable to create a security context for the session.

When the certificate store is searched for a matching certificate, it will first search for any certificate with a friendly name that matches the property value. If no valid certificate is found, it will then search for a certificate with a matching common name.

Certificates may be installed and viewed on the local system using the Certificate Manager that is included with the Windows operating system. For more information, refer to the documentation for the Microsoft Management Console.

See Also

InternetMail Class | SocketTools Namespace | CertificateStore Property | Secure Property

InternetMail.CertificateStatus Property

Gets a value which indicates the status of the security certificate returned by the remote host.

[Visual Basic]

```
Public ReadOnly Property CertificateStatus As SecurityCertificate
```

[C#]

```
public InternetMail.SecurityCertificate CertificateStatus {get;}
```

Property Value

A SecurityCertificate enumeration value which specifies the status of the certificate.

Remarks

The **CertificateStatus** property is used to determine the status of the security certificate returned by the remote host when a secure connection has been established. This property value should be checked after the connection to the server has completed, but prior to beginning a transaction.

Note that if the certificate cannot be validated, the secure connection will not be automatically terminated. It is the responsibility of your application to determine the best course of action to take if the certificate is invalid. Even if the security certificate cannot be validated, the data exchanged with the remote host will still be encrypted.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.CertificateStore Property

Gets and sets a value that specifies the name of the local certificate store.

[Visual Basic]

```
Public Property CertificateStore As String
```

[C#]

```
public string CertificateStore {get; set;}
```

Property Value

A string which specifies the certificate store name. The default value is the current user's personal certificate store.

Remarks

The **CertificateStore** property is used to specify the name of the certificate store which contains the security certificate to use when establishing a secure connection. The certificate may either be stored in the registry or in a file. If the certificate is stored in the registry, then this property should be set to one of the following predefined values:

Store Name	Description
CA	Certification authority certificates. These are certificates that are issued by entities which are entrusted to issue certificates to other individuals or organizations. Companies such as VeriSign and Thawte act as certification authorities.
MY	Personal certificates and their associated private keys for the current user. This store typically holds the client certificates used to establish a user's credentials. If a certificate store is not specified, this is the default value that is used.
ROOT	Certificates that have been self-signed by a certificate authority. Root certificates for a number of different certification authorities such as VeriSign and Thawte are installed as part of the operating system and periodically updated by Microsoft.

In most cases the client certificate will be installed in the user's personal certificate store, and therefore it is not necessary to set this property value because that is the default location that will be used to search for the certificate. This property is only used if the **CertificateName** property is also set to a valid certificate name.

If you are using a local certificate store, with the certificate and private key stored in the registry, you can explicitly specify whether the certificate store for the current user or the local machine (all users) should be used. This is done by prefixing the certificate store name with "HKCU" for the current user, or "HKLM" for the local machine. For example, a certificate store name of "HKLM:MY" would specify the personal certificate store for the local machine, rather than the current user. If neither prefix is specified, it will default to the certificate store for the current user.

This property may also be used to specify a file that contains the client certificate. In this case, the property should specify the full path to the file and must contain both the certificate and private key in PKCS12

format. If the file is protected by a password, the **CertificatePassword** property must also be set to specify the password.

See Also

InternetMail Class | SocketTools Namespace | CertificatePassword Property | Secure Property

InternetMail.CertificateSubject Property

Gets a value that provides information about the organization that the server certificate was issued to.

[Visual Basic]

```
Public ReadOnly Property CertificateSubject As String
```

[C#]

```
public string CertificateSubject {get;}
```

Property Value

A string that contains a comma separated list of name value pairs.

Remarks

The **CertificateSubject** property returns a string that contains information about the organization that the server certificate was issued to. The string value is a comma separated list of tagged name and value pairs. In the nomenclature of the X.500 standard, each of these pairs are called a relative distinguished name (RDN), and when concatenated together, forms the issuer's distinguished name (DN). For example:

```
C=US, O="RSA Data Security, Inc.", OU=Secure Server Certification Authority
```

To obtain a specific value, such as the name of the issuer or the issuer's country, the application must parse the string returned by this property. Some of the common tokens used in the distinguished name are:

Token	Description
C	The ISO standard two character country code.
S	The name of the state or province.
L	The name of the city or locality.
O	The name of the company or organization.
OU	The name of the department or organizational unit
CN	The common name; with X.509 certificates, this is the domain name of the site the certificate was issued for.

This property will return an empty string if a secure connection has not been established with the remote host.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.CipherStrength Property

Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.

[Visual Basic]

```
Public ReadOnly Property CipherStrength As Integer
```

[C#]

```
public int CipherStrength {get;}
```

Property Value

An integer value which specifies the encryption key length if a secure connection has been established; otherwise a value of 0 is returned.

Remarks

The **CipherStrength** property returns the number of bits in the key used to encrypt the secure data stream. Common values returned by this property are 128 and 256. A key length of 40 or 56 bits is considered insecure and subject to brute force attacks. 128-bit and 256-bit keys are considered secure. If this property returns a value of 0, this means that a secure connection has not been established with the remote host.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.ContentId Property

Gets the content identifier for the current message part.

[Visual Basic]
```
Public ReadOnly Property ContentId As String
```

[C#]
```
public string ContentId {get;}
```

Property Value

A string which specifies the content identifier.

Remarks

The **ContentId** property returns the unique content identifier string for the current message part. This multipart header field is not commonly used, and if undefined, will return an empty string.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.ContentLength Property

Gets the size of the data stored in the current message part.

[Visual Basic]

```
Public ReadOnly Property ContentLength As Integer
```

[C#]

```
public int ContentLength {get;}
```

Property Value

An integer which specifies the size of the current message part in bytes.

Remarks

The **ContentLength** property returns the size of the data stored in the selected message part. This property is read-only, and is updated when the current message part changes.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.ContentType Property

Gets and sets the content type of the selected message part.

[Visual Basic]
```
Public Property ContentType As String
```

[C#]
```
public string ContentType {get; set;}
```

Property Value

A string which specifies the content type.

Remarks

The **ContentType** property returns the MIME type for the currently selected message part. The type string consists of a primary type and secondary sub-type separated by a slash, followed by one or more optional parameters delimited by semi-colons. For example, this is a common content type for text messages:

```
text/plain; charset=utf-8
```

The **text** designation indicates that this message part contains readable text, and the **plain** sub-type indicates that the text does not contain any special encoding. The optional parameter which follows the content type provides additional information about the content. In this example, it specifies which character set should be used to display the text. The two common character sets used are UTF-8 and US-ASCII.

There are seven predefined, standard content types, each with their own sub-types. The following table lists these types, along with some common sub-types that are found in messages:

Type	Description
text	Indicates that the message part contains text. This is the most common type found in mail messages; if no content type is explicitly defined, then it is assumed to be plain text. Examples are text/plain, text/richtext and text/html.
image	Indicates that the message part contains a graphics image. Examples are image/gif and image/jpeg.
audio	Indicates that the message part contains audio data; the basic sub-type is 8-bit PCM encoded audio (commonly found with the .au filename extension). Examples are audio/basic, audio/aiff and audio/wav.
video	Indicates that the message part contains a video clip in the specified format. Examples are video/mpeg and video/avi.
application	Indicates that the message part contains application specific data, typically used with the octet-stream sub-type to indicate binary file attachments for executable programs, compressed

	file archives, etc. Examples are application/octet-stream and application/postscript.
message	Indicates that the message part contains a complete RFC 822 compliant message, complete with headers. An example is message/rfc822.
multipart	Indicates that this is part of a mixed message (a message that contains multiple parts of different content types). Examples are multipart/alternative and multipart/mixed.

The three most common content types that are used in applications are text/plain for the mail message body, application/octet-stream for binary file attachments and multipart/mixed for messages that contain both text and attached files. For more information about the different content types, refer to the Multipurpose Internet Mail Extensions (MIME) standards document RFC 1521.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Date Property

Gets and sets the date for the current message.

[Visual Basic]
```
Public Property Date As String
```

[C#]
```
public string Date {get; set;}
```

Property Value

A string which specifies the date in RFC 822 format.

Remarks

The **Date** property returns the value of the date field in the current message header. Setting this property causes the date field to be updated with the specified value. When setting the date, any one of the following formats may be used:

Format	Example
mm/dd/yy[yy] hh:mm[:ss]	03/01/2006 12:00:00
yy[yy]/mm/dd hh:mm[:ss]	2006/03/01 12:00:00
dd mmm yy[yy] hh:mm[:ss]	01 Mar 2006 12:00:00
mmm dd yy[yy] hh:mm[:ss]	Mar 01 2006 12:00:00

Any extraneous information that may be included in the date string, such as the day of the week, is ignored. In addition to the date and time, the string may also include a time zone specification at the end. If no time zone is specified, the current time zone is used.

When specifying a time zone, the value should either be prefixed by a plus sign (+) to indicate that the time zone is to the east of GMT, or a minus sign (-) to indicates that it's to the west. Four digits follow, with the first two indicating the number of hours east or west of GMT, and the last two digits indicating the number of minutes. Therefore, a value of -0800 means that the time zone is eight hours to the west of GMT, or in other words, the Pacific time zone.

Regardless of the format of the string assigned to the property, it always returns the date in the same format (which conforms to the RFC 822 specification). Using the above examples, the date would be returned as "Wed, 01 Mar 2006 12:00:00 -0800" if you are located in the Pacific timezone.

The **Localize** property affects how dates are processed by the control. If enabled, dates are automatically adjusted for the local time zone. By default, localization is disabled.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Delivered Property

Gets the number of recipients the message has been delivered to.

[Visual Basic]

```
Public ReadOnly Property Delivered As Integer
```

[C#]

```
public int Delivered {get;}
```

Property Value

An integer value which specifies the number of recipients the message was successfully delivered to.

Remarks

The **Delivered** property returns the number of recipients that the message was successfully delivered to using the **SendMessage** method. This property is updated after the method returns, and can be compared against the value of the **Recipients** property to check that the message was submitted to all recipients.

It is important to note that there is no guarantee that the recipient has retrieved and read the message, only that it has been submitted successfully to the mail server for delivery.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Domain Property

Gets and sets the local domain name.

[Visual Basic]

```
Public Property Domain As String
```

[C#]

```
public string Domain {get; set;}
```

Property Value

A string value which specifies the local domain name.

Remarks

The **Domain** property specifies the domain name of the local host, and is used to identify the current system when sending messages. If this property is not defined, then the local host name will be used.

Note that explicitly setting the **Domain** property to a value that does not match your local host name may cause some mail servers to reject any messages that you attempt to send.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Encoding Property

Gets and sets the content encoding method used for the current message part.

[Visual Basic]

```
Public Property Encoding As String
```

[C#]

```
public string Encoding {get; set;}
```

Property Value

A string which specifies the encoding type.

Remarks

The **Encoding** property returns the method used for encoding the current message part. Setting this property causes the Content-Transfer-Encoding header value to be updated. The following values are commonly used:

Type	Description
7bit	The default transfer encoding type, which consists of printable ASCII characters.
8bit	Printable ASCII characters, including those characters with the high-bit set as is common with the ISO Latin-1 character set.
binary	All characters; binary transfer encoding is rarely used.
quoted-printable	Printable ASCII characters, with non-printable or extended characters represented using their hexadecimal equivalents.
base64	The transfer encoding type commonly used to convert binary data into 7-bit ASCII characters so that it may be transported safely through the mail system.
x-uuencode	A transfer encoding type similar in function to the base64 encoding method.

Note that setting this property only updates the Content-Transfer-Encoding header value. To control the actual encoding method used for attachments, specify the encoding method when calling the **AttachFile** method.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.From Property

Gets and sets the address of the user who sent the message.

[Visual Basic]
```
Public Property From As String
```

[C#]
```
public string From {get; set;}
```

Property Value

A string which specifies the sender of the message.

Remarks

The **From** property returns the address of the user who sent the message. Setting the property causes the From header field to be updated with the new value.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.HashStrength Property

Gets a value which specifies the length of the message digest that was selected for a secure connection.

[Visual Basic]

```
Public ReadOnly Property HashStrength As Integer
```

[C#]

```
public int HashStrength {get;}
```

Property Value

An integer value which specifies the length of the message digest if a secure connection has been established; otherwise a value of 0 is returned.

Remarks

The **HashStrength** property returns the number of bits used in the message digest (hash) that was selected. Common values returned by this property are 128 and 160. If this property returns a value of 0, this means that a secure connection has not been established with the remote host.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.IsConnected Property

Gets a value which indicates if a connection to the mail server has been established.

[Visual Basic]

```
Public ReadOnly Property IsConnected As Boolean
```

[C#]

```
public bool IsConnected {get;}
```

Property Value

Returns **true** if the connection has been established; otherwise returns **false**.

Remarks

The **IsConnected** property can only be used to indicate if there is still a logical connection to the mail server. It cannot be used to detect abnormal conditions such as the remote host aborting the connection, the physical network connection being lost or other critical errors.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.IsIdle Property

Gets a value which indicates if the client is idle and the current mailbox is being monitored for status changes.

[Visual Basic]
```
Public ReadOnly Property IsIdle As Boolean
```

[C#]
```
public bool IsIdle {get;}
```

Property Value

Returns **true** if the client is idle and the current mailbox is being monitored; otherwise returns **false**.

Remarks

The **IsIdle** property can be used to determine if the **Idle** method has been called to place the client session in an idle state, monitoring the connection for any status messages sent by the server. Typically this is done to allow the application to be notified asynchronously whenever a new message is stored in the mailbox, or when a message has been expunged.

The worker thread that monitors the client connection in the background can terminate if an IMAP command is sent to the server, if the **Cancel** method is called or if the client disconnects from the server. This property enables the application to determine if this background thread is still active or not.

See Also

InternetMail Class | SocketTools Namespace | Idle Method | OnUpdate Event

InternetMail.IsInitialized Property

Gets a value which indicates if the current instance of the class has been initialized successfully.

[Visual Basic]

```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]

```
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the class instance has been initialized; otherwise returns **false**.

Remarks

The **IsInitialized** property is used to determine if the current instance of the class has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the class may not be able to initialize itself.

The most common reasons that a class instance may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required libraries or not being able to access the system registry.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public InternetMail.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.LastMessage Property

Gets the number of the last message available in the current mailbox.

[Visual Basic]

```
Public ReadOnly Property LastMessage As Integer
```

[C#]

```
public int LastMessage {get;}
```

Property Value

An integer value which specifies the number of messages.

Remarks

The **LastMessage** property returns the number of the last message available in the currently selected mailbox. This value may be different than the value returned by the **MessageCount** property if one or more messages have been deleted. This is because the **MessageCount** property returns the number of available messages, and this value will decrement whenever a message is flagged for deletion. The **LastMessage** property value is constant and will always return the last message number in the mailbox, regardless if any messages have been deleted.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.LocalAddress Property

Gets and sets the local Internet address that the client will be bound to.

[Visual Basic]

```
Public ReadOnly Property LocalAddress As String
```

[C#]

```
public string LocalAddress {get;}
```

Property Value

A string which specifies an Internet address in dotted notation.

Remarks

The **LocalAddress** property is used to specify the local Internet address that the client will be bound to when a connection is established with a remote host. By default this property is not assigned a value, which specifies that the client should be bound to any appropriate network interface on the local system.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Localize Property

Gets and sets a value which specifies if date and time values should be localized.

[Visual Basic]

```
Public Property Localize As Boolean
```

[C#]

```
public bool Localize {get; set;}
```

Property Value

A boolean value which specifies if date and time values should be localized.

Remarks

The **Localize** property is used to enable or disable localization features of the class. Currently this only affects the way in which dates are processed by the class. If set to **true**, the control will adjust for the local time zone when setting and reading the **Date** property. The default value for this property is **false**.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.LocalName Property

Gets a value which specifies the host name for the local system.

[Visual Basic]

```
Public ReadOnly Property LocalName As String
```

[C#]

```
public string LocalName {get;}
```

Property Value

A string which specifies the local host name.

Remarks

The **LocalName** property returns the fully-qualified host name assigned to the local system. If the system has not been configured with an Internet domain name, then this property will return the NetBIOS name assigned to the local system.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.MailboxArray Class

The class used to return the available mailboxes for the current user.

For a list of all members of this type, see InternetMail.MailboxArray Members.

System.Object
  SocketTools.InternetMail.MailboxArray

[Visual Basic]

```
<DefaultMember(MemberName:="Item")>
Public Class InternetMail.MailboxArray
```

[C#]

```
[DefaultMember(MemberName="Item")]
public class InternetMail.MailboxArray
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The **MailboxArray** class returns the available mailboxes for the current user. A read-only instance of this class is created by the class constructor and accessed using the **Mailbox** array.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

InternetMail.MailboxArray Members | SocketTools Namespace

InternetMail.MailboxArray Members

InternetMail.MailboxArray overview

Public Instance Properties

Count	Returns the maximum number of elements in the array.
Item	Gets the name of a mailbox.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetMail.MailboxArray Class | SocketTools Namespace

InternetMail.MailboxArray Properties

The properties of the **InternetMail.MailboxArray** class are listed below. For a complete list of **InternetMail.MailboxArray** class members, see the InternetMail.MailboxArray Members topic.

Public Instance Properties

Count	Returns the maximum number of elements in the array.
Item	Gets the name of a mailbox.

See Also

InternetMail.MailboxArray Class | SocketTools Namespace

InternetMail.MailboxArray.Count Property

Returns the maximum number of elements in the array.

[Visual Basic]
```
Public ReadOnly Property Count As Integer
```

[C#]
```
public int Count {get;}
```

Property Value

An integer value.

Remarks

This property will return the same value as the **Mailboxes** property and is used to determine the maximum index value for the **Mailbox** array.

See Also

InternetMail.MailboxArray Class | SocketTools Namespace

InternetMail.MailboxArray.Item Property

Gets the name of a mailbox.

[Visual Basic]

```
Public Default ReadOnly Property Item( _
   ByVal index As Integer _
) As String
```

[C#]

```
public string this[
   int index
] {get;}
```

Parameters

index

An integer value which specifies the index into the array.

Property Value

A string which specifies the name of a mailbox.

See Also

InternetMail.MailboxArray Class | SocketTools Namespace

InternetMail.Mailboxes Property

Gets the number of mailboxes available on the server.

[Visual Basic]

```
Public ReadOnly Property Mailboxes As Integer
```

[C#]

```
public int Mailboxes {get;}
```

Property Value

An integer value which specifies the number of available mailboxes.

Remarks

The **Mailboxes** property returns the total number of mailboxes available to the current account on the server. This property can be used in conjunction with the **Mailbox** array to enumerate the names of all of the mailboxes which can be selected by the client.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.MailboxFlags Property

Gets one or more flags which identify characteristics of the current mailbox.

[Visual Basic]

```
Public ReadOnly Property MailboxFlags As ImapFlags
```

[C#]

```
public InternetMail.ImapFlags MailboxFlags {get;}
```

Property Value

An ImapFlags enumeration value which specifies one or more mailbox flags.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.MailboxMask Property

Gets and sets the current mailbox wildcard mask.

[Visual Basic]

```
Public Property MailboxMask As String
```

[C#]

```
public string MailboxMask {get; set;}
```

Property Value

A string which specifies the current mailbox wildcard mask.

Remarks

The **MailboxMask** property returns the current mailbox wildcard mask. If no wildcard mask has been specified by the client, this property will return an empty string.

Setting the **MailboxMask** property will determine which mailboxes are returned by the **Mailbox** array. Wildcards may include the asterisk (which matches any mailbox as well as any child mailboxes) and the percent sign (which matches any mailbox, but does not match any child mailboxes). This property may be used in conjunction with the **MailboxPath** property to further qualify which mailboxes are returned.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.MailboxName Property

Gets and sets the name of the current mailbox.

[Visual Basic]
```
Public Property MailboxName As String
```

[C#]
```
public string MailboxName {get; set;}
```

Property Value

A string that specifies the name of the current mailbox.

Remarks

The **MailboxName** property returns the name of the currently selected mailbox. If no mailbox has been selected by the client, this property will return an empty string.

Setting the **MailboxName** property will select a new mailbox in read-write mode. If the client has a different mailbox currently selected, that mailbox will be closed and any messages marked for deletion will be expunged. To prevent deleted messages from being removed from the previous mailbox, call the **UnselectMailbox** method prior to selecting the new mailbox. Setting the **MailboxName** property to an empty string will cause the current mailbox to be unselected, and a new mailbox will not be selected. Before the application can access any messages, it must select a new mailbox.

Selecting a new mailbox will automatically update those properties which provide information about the current mailbox, such as the **MailboxFlags** and **MailboxUID** properties. If an application wishes to update the information for the current mailbox, simply set the **MailboxName** property again with the same mailbox name. Note that this will not cause any messages marked for deletion to be expunged.

The special case-insensitive mailbox name INBOX is used for new messages. Other mailbox names may or may not be case-sensitive depending on the IMAP server's operating system and implementation.

If the mailbox name contains international characters then it is automatically encoded using a modified version of UTF-7 encoding. For example, if a mailbox is named "Håndskrift", the mailbox name created on the server will be the string "H&AOU-ndskrift". The control will automatically decode UTF-7 encoded mailbox names, making the conversion transparent to the application.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.MailboxPath Property

Gets and sets the current mailbox reference path.

[Visual Basic]

```
Public Property MailboxPath As String
```

[C#]

```
public string MailboxPath {get; set;}
```

Property Value

A string which specifies the current mailbox reference path.

Remarks

The **MailboxPath** property returns the current mailbox reference path. If no path has been specified by the client, this property will return an empty string.

Setting the **MailboxPath** property will determine which mailboxes are returned by the **Mailbox** array. Typically this is used to specify a subdirectory where mail folders are stored for the current user. Note that some mail servers may not permit absolute reference paths, and in most cases the path should include a trailing slash. This property may be used in conjunction with the **MailboxMask** property to further qualify which mailboxes are returned.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.MailboxSize Property

Gets the size of the current mailbox.

[Visual Basic]
```
Public ReadOnly Property MailboxSize As Integer
```

[C#]
```
public int MailboxSize {get;}
```

Property Value

An integer value which specifies the size of the mailbox in bytes.

Remarks

The **MailboxSize** property returns the combined size of all messages in the current mailbox. Referencing this property will cause the current thread to block and may require a significant amount of time to calculate the mailbox size if there are a large number of messages in the mailbox. Because it can potentially result in long delays, it is not recommended that an application calculate the mailbox size unless it is absolutely necessary.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.MailboxUID Property

Gets the unique identifier for the current mailbox.

[Visual Basic]
```
Public ReadOnly Property MailboxUID As Integer
```

[C#]
```
public int MailboxUID {get;}
```

Property Value

An integer value which specifies the mailbox UID.

Remarks

The **MailboxUID** property returns an integer value which uniquely identifies the mailbox and corresponds to the UIDVALIDITY value returned by the IMAP server. The actual value is determined by the server and should be considered opaque. The protocol specification requires that a mailbox's UID must not change unless the mailbox contents are modified or existing messages in the mailbox have been assigned new UIDs.

An application can use the **MailboxUID** property value in combination with the **MessageUID** property in order to uniquely identify a message on the server. However, the application must take into consideration that the IMAP server can reassign new message UIDs when the mailbox is modified. If the mailbox and message UIDs are being stored on the local system to track what messages have been retrieved from the server, the application must check the UID of the mailbox whenever it is selected. If the mailbox UID has changed, this means that the UIDs for the messages in that mailbox may have changed. The client should resynchronize with the server, and update it's local copy of that mailbox.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Mailer Property

Gets and sets the name of the mailer application.

[Visual Basic]

```
Public Property Mailer As String
```

[C#]

```
public string Mailer {get; set;}
```

Property Value

A string which specifies the name of the mailer application.

Remarks

The **Mailer** property returns the value of the X-Mailer field in the current message header. Setting this property causes the field to be updated with the specified value. This is typically used to identify the program which generated the message.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Message Property

Gets and sets the current message headers and body.

[Visual Basic]

```
Public Property Message As String
```

[C#]

```
public string Message {get; set;}
```

Property Value

A string which contains the complete message.

Remarks

The **Message** property returns the current message, including the headers and all message parts, as a string. Setting this property will cause the current message to be cleared and replaced by the new value. The contents must follow the standard specifications for a message. If the property is set to an empty string, the current message is cleared.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.MessageCount Property

Gets the number of messages available in the current mailbox.

[Visual Basic]

```
Public ReadOnly Property MessageCount As Integer
```

[C#]

```
public int MessageCount {get;}
```

Property Value

An integer value which specifies the number of messages.

Remarks

The **MessageCount** property returns the number of messages available to be retrieved from the currently selected mailbox.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.MessageFlags Property

Gets and sets one or more flags for the current message.

[Visual Basic]

```
Public Property MessageFlags As ImapFlags
```

[C#]

```
public InternetMail.ImapFlags MessageFlags {get; set;}
```

Property Value

An ImapFlags enumeration value which specifies one or more message flags.

Remarks

The **MessageFlags** property returns information about the currently selected message specified by the **Message** property. Setting the **MessageFlags** property changes the flags for the currently selected message. Multiple bit flags can be combined using the bitwise Or operator. An application can test if a flag is set by using the bitwise And operator.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.MessageID Property

Gets the current message identifier.

[Visual Basic]

```
Public Property MessageID As String
```

[C#]

```
public string MessageID {get; set;}
```

Property Value

A string which specifies the message identifier.

Remarks

The **MessageID** property returns a unique string that can be used to identify the message.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.MessageIndex Property

Gets and sets the current message number.

[Visual Basic]

```
Public Property MessageIndex As Integer
```

[C#]

```
public int MessageIndex {get; set;}
```

Property Value

An integer value which specifies the current message number.

Remarks

The **Message** property sets or returns the message number for the currently selected mailbox. Message numbers range from 1 through the number of messages available on the server, as returned by the **MessageCount** property. Setting the **Message** property to an invalid message number will generate an exception.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.MessagePart Property

Gets and sets the current section index in a multipart message.

[Visual Basic]

```
Public Property MessagePart As Integer
```

[C#]

```
public int MessagePart {get; set;}
```

Property Value

An integer which specifies the current message part.

Remarks

The **MessagePart** property returns the current message part index. All messages have at least one part, which consists of one or more header fields, followed by the body of the message. The default part, part 0, refers to the main message header and body. If the message contains multiple parts (as with a message that contains one or more attached files), the **Part** property can be set to refer to that specific part of the message.

For example, messages with file attachments typically consist of a message part which describes the contents of the attachment, followed by the attachment itself. For a message with one attached file, there would be a total of three parts. Part 0 would refer to the main message part, which contains the headers such as From, To, Subject, Date and so on. For multipart messages, part 0 typically does not have a message body, since any text is usually created as a separate part (for those messages that do not contain multiple parts, the part 0 body contains the text message). Part 1 would contain the text describing the attachment, and part 2 would contain the attachment itself. If the attached file is binary, then the transfer encoding type would usually be base64.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.MessageParts Property

Gets the number of sections in a multipart message.

[Visual Basic]

```
Public ReadOnly Property MessageParts As Integer
```

[C#]

```
public int MessageParts {get;}
```

Property Value

An integer value which specifies the number of message parts.

Remarks

The **MessageParts** property returns the number of parts in the current message. All messages have at least one part, referenced as part 0. Multipart messages will consist of additional parts which may be accessed by setting the **MessagePart** property.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.MessageSize Property

Gets the size of the current message in bytes.

[Visual Basic]
```
Public ReadOnly Property MessageSize As Integer
```

[C#]
```
public int MessageSize {get;}
```

Property Value

An integer value which specifies the size of the message.

Remarks

The **MessageSize** property returns the size of the current message in bytes. The size includes the header and body portion of the message.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.MessageText Property

Gets and sets the text body of the current message part.

[Visual Basic]

```
Public Property MessageText As String
```

[C#]

```
public string MessageText {get; set;}
```

Property Value

A string which contains the body of the current message part.

Remarks

The **MessageText** property returns the body of the current message part. Setting this property replaces the body of the current message part with the new text.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.MessageUID Property

Gets the UID for the current message.

[Visual Basic]

```
Public ReadOnly Property MessageUID As String
```

[C#]

```
public string MessageUID {get;}
```

Property Value

An string value which specifies the current message UID.

Remarks

The **MessageID** property returns a string which uniquely identifies the message on the server. The identifier is assigned by the mail server, and retains the same value across multiple client sessions. This value is typically used when the client wants to leave a message on the mail server, but does not wish to retrieve the message contents multiple times. For example, the client can store the unique ID for each message that it retrieves, but does not delete from the server. The next time that it connects to the mail server, it compares the unique ID of a message against the stored values. If there is a match, the client knows that the message has already been retrieved, and does not need to do so again.

This property requires that the server support the optional UIDL command. If the command is not supported, this property will always return an empty string. Note that the unique ID for the message is not the same as the Message-ID header field in the message itself.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.MimeVersion Property

Gets and sets the MIME version number for the current message.

[Visual Basic]

```
Public Property MimeVersion As String
```

[C#]

```
public string MimeVersion {get; set;}
```

Property Value

A string that specifies the version number.

Remarks

The **MimeVersion** property returns the version number for the current message. Setting this property causes the MIME-Version header value to be changed to the specified value. An empty string causes the MIME version number to be set to the default value of "1.0". It is recommended that you do not change the value of this property.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.NameServerArray Class

The class used to return the nameservers configured for the local host.

For a list of all members of this type, see InternetMail.NameServerArray Members.

System.Object
  SocketTools.InternetMail.NameServerArray

[Visual Basic]
```
<DefaultMember(MemberName:="Item")>
Public Class InternetMail.NameServerArray
```

[C#]
```
[DefaultMember(MemberName="Item")]
public class InternetMail.NameServerArray
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The **NameServerArray** class returns the nameservers configured for the local host. A read-only instance of this class is created by the InternetMail class constructor and accessed using the **NameServer** array.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

InternetMail.NameServerArray Members | SocketTools Namespace | NameServer Field

InternetMail.NameServerArray Members

InternetMail.NameServerArray overview

Public Instance Properties

Count	Returns the maximum number of elements in the array.
Item	Returns the Internet address for the specified nameserver.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetMail.NameServerArray Class | SocketTools Namespace | NameServer Field

InternetMail.NameServerArray Properties

The properties of the **InternetMail.NameServerArray** class are listed below. For a complete list of **InternetMail.NameServerArray** class members, see the InternetMail.NameServerArray Members topic.

Public Instance Properties

Count	Returns the maximum number of elements in the array.
Item	Returns the Internet address for the specified nameserver.

See Also

InternetMail.NameServerArray Class | SocketTools Namespace | NameServer Field

InternetMail.NameServerArray.Count Property

Returns the maximum number of elements in the array.

[Visual Basic]

```
Public ReadOnly Property Count As Integer
```

[C#]

```
public int Count {get;}
```

Property Value

An integer value.

Remarks

This property will return the maximum number of nameservers that may be configured.

See Also

InternetMail.NameServerArray Class | SocketTools Namespace

InternetMail.NameServerArray.Item Property

Returns the Internet address for the specified nameserver.

[Visual Basic]

```
Public Default Property Item( _
   ByVal index As Integer _
) As String
```

[C#]

```
public string this[
   int index
] {get; set;}
```

Parameters

index

An integer value which specifies the index into the array.

Property Value

A string which specifies an Internet address using dot notation.

See Also

InternetMail.NameServerArray Class | SocketTools Namespace

InternetMail.NewMessages Property

Gets the number of new messages available in the current mailbox.

[Visual Basic]

```
Public ReadOnly Property NewMessages As Integer
```

[C#]

```
public int NewMessages {get;}
```

Property Value

An integer value which specifies the number of new, unread messages in the current mailbox.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Options Property

Gets and sets a value which specifies one or more client options.

[Visual Basic]
```
Public Property Options As InternetMailOptions
```

[C#]
```
public InternetMail.InternetMailOptions Options {get; set;}
```

Property Value

Returns one or more InternetMailOptions enumeration flags which specify the options for the client. The default value for this property is **optionNone**.

Remarks

The **Options** property specifies one or more default options options which are used when establishing a connection using the **Connect** method.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Organization Property

Gets and sets the name of the organization that originated the message.

[Visual Basic]

```
Public Property Organization As String
```

[C#]

```
public string Organization {get; set;}
```

Property Value

A string which specifies the organization name.

Remarks

The **Organization** property returns the name of the organization that originated the current message. Setting this property updates the specified header value. Note that many mail clients do not generate an Organization header field, in which case the property value will be an empty string.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Password Property

Gets and sets the password used to authenticate the client.

[Visual Basic]

```
Public Property Password As String
```

[C#]

```
public string Password {get; set;}
```

Property Value

A string which specifies the password.

Remarks

The **Password** property specifies the password used to authenticate the client session. This property is used as the default value for the **Connect** method if no password is specified as an argument.

Assigning a value to his property will clear any OAuth 2.0 bearer token which may have been previously assigned to the **BearerToken** property and default authentication methods will be used with the mail service.

If the **BearerToken** property has been set, the client will use OAuth 2.0 for authentication and this property will return an empty string.

See Also

InternetMail Class | SocketTools Namespace | BearerToken Property | UserName Property | Connect Method

InternetMail.Priority Property

Gets and sets the current message priority.

[Visual Basic]

```
Public Property Priority As String
```

[C#]

```
public string Priority {get; set;}
```

Property Value

A string which specifies the message priority.

Remarks

The **Priority** property returns the current priority for the message. Setting this property value causes the X-Priority header to be updated with the specified value.

There is no standard for specifying message priority. The convention is to use a number from 1-5, with 1 indicating the highest priority, 3 as normal priority and 5 as the lowest priority. Some mailers follow the number with a space and then text that describes the priority level.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.RecentMessages Property

Gets the number of messages which have recently arrived in the mailbox.

[Visual Basic]

```
Public ReadOnly Property RecentMessages As Integer
```

[C#]

```
public int RecentMessages {get;}
```

Property Value

An integer value which specifies the number of recent messages.

Remarks

The **RecentMessages** property returns the number of messages which have been recently added to the currently selected mailbox. This property is particularly useful when the INBOX mailbox is selected because it enables the application to check if any new messages have arrived.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.RecipientArray Class

The class used to return the recipient addresses for the current message.

For a list of all members of this type, see InternetMail.RecipientArray Members.

System.Object
 SocketTools.InternetMail.RecipientArray

[Visual Basic]
```
<DefaultMember(MemberName:="Item")>
Public Class InternetMail.RecipientArray
```

[C#]
```
[DefaultMember(MemberName="Item")]
public class InternetMail.RecipientArray
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The **RecipientArray** class returns the recipient addresses for the current message. A read-only instance of this class is created by the InternetMail class constructor and accessed using the **Recipient** array.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

InternetMail.RecipientArray Members | SocketTools Namespace

InternetMail.RecipientArray Members

InternetMail.RecipientArray overview

Public Instance Properties

Count	Returns the maximum number of elements in the array.
Item	Gets the email address of a recipient specified in the current message.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Gets the recipient addresses for the current message.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetMail.RecipientArray Class | SocketTools Namespace

InternetMail.RecipientArray Properties

The properties of the **InternetMail.RecipientArray** class are listed below. For a complete list of **InternetMail.RecipientArray** class members, see the InternetMail.RecipientArray Members topic.

Public Instance Properties

Count	Returns the maximum number of elements in the array.
Item	Gets the email address of a recipient specified in the current message.

See Also

InternetMail.RecipientArray Class | SocketTools Namespace

InternetMail.RecipientArray.Count Property

Returns the maximum number of elements in the array.

[Visual Basic]
```
Public ReadOnly Property Count As Integer
```

[C#]
```
public int Count {get;}
```

Property Value

An integer value.

Remarks

This property will return the same value as the **Recipients** property and is used to determine the maximum index value for the **Recipient** array.

See Also

InternetMail.RecipientArray Class | SocketTools Namespace

InternetMail.RecipientArray.Item Property

Gets the email address of a recipient specified in the current message.

[Visual Basic]

```
Public Default ReadOnly Property Item( _
   ByVal index As Integer _
) As String
```

[C#]

```
public string this[
   int index
] {get;}
```

Parameters

index

An integer value which specifies the index into the array.

Property Value

A string which contains an email address.

See Also

InternetMail.RecipientArray Class | SocketTools Namespace

InternetMail.RecipientArray Methods

The methods of the **InternetMail.RecipientArray** class are listed below. For a complete list of **InternetMail.RecipientArray** class members, see the InternetMail.RecipientArray Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Gets the recipient addresses for the current message.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetMail.RecipientArray Class | SocketTools Namespace

InternetMail.RecipientArray.ToString Method

Gets the recipient addresses for the current message.

[Visual Basic]

```
Overrides Public Function ToString() As String
```

[C#]

```
public override string ToString();
```

Return Value

A string which specifies the email addresses for the recipients of the current message.

Remarks

The **ToString** method returns a comma separated list of email addresses for all recipients specified in the current message.

See Also

InternetMail.RecipientArray Class | SocketTools Namespace

InternetMail.Recipients Property

Gets the number of recipients specified in the current message.

[Visual Basic]

```
Public ReadOnly Property Recipients As Integer
```

[C#]

```
public int Recipients {get;}
```

Property Value

An integer which specifies the number of recipients.

Remarks

The **Recipients** property returns the number of recipient addresses that have been specified in the current message. This includes all of the addresses listed in the To, Cc and Bcc header fields. This property can be used in conjunction with the **Recipient** array to enumerate all of the recipient addresses in the message.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.RelayPort Property

Gets and sets a value which specifies the relay server port number.

[Visual Basic]

```
Public Property RelayPort As Integer
```

[C#]

```
public int RelayPort {get; set;}
```

Property Value

An integer value which specifies a port number.

Remarks

The **RelayPort** property is used to set the port number that will be used to establish a connection with a relay server when sending a message.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.RelayServer Property

Gets and sets a value which specifies the relay server name or address.

[Visual Basic]

```
Public Property RelayServer As String
```

[C#]

```
public string RelayServer {get; set;}
```

Property Value

A string which specifies a host name.

Remarks

The **RelayServer** property can be used to set the host name for the relay mail server that will be responsible for delivering the message.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.ReplyTo Property

Gets and sets the address of the user who should receive replies to this message.

[Visual Basic]

```
Public Property ReplyTo As String
```

[C#]

```
public string ReplyTo {get; set;}
```

Property Value

A string that specifies an email address.

Remarks

The **ReplyTo** property returns the address of the user who should receive replies to the current message. Setting this property updates the Reply-To header with the specified value.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.ReturnReceipt Property

Gets and sets the address of the person who should receive a message indicating that the message has been read.

[Visual Basic]

```
Public Property ReturnReceipt As String
```

[C#]

```
public string ReturnReceipt {get; set;}
```

Property Value

A string value which specifies an email address.

Remarks

The **ReturnReceipt** property returns the address of the person who should receive a message indicating that the current message has been read. Setting this property updates the Disposition-Notification-To header field with the specified value.

Setting the **ReturnReceipt** property does not automatically cause an acknowledgement to be returned to the sender. An application is responsible for checking to make sure the header field contains a valid address and then generating the return receipt message.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Secure Property

Gets and sets a value which specifies if a secure connection is established.

[Visual Basic]

```
Public Property Secure As Boolean
```

[C#]

```
public bool Secure {get; set;}
```

Property Value

Returns **true** if a secure connection is established; otherwise returns **false**. The default value is **false**.

Remarks

The **Secure** property determines if a secure connection is established with the remote host. The default value for this property is **false**, which specifies that a standard connection to the server is used. To establish a secure connection, the application should set this property value to **true** prior to calling the **Connect** method. Once the connection has been established, the client may exchange data with the server as with standard connections.

It is strongly recommended that any application that sets this property **true** use error handling to trap an errors that may occur. If the control is unable to initialize the security libraries, or otherwise cannot create a secure session for the client, an exception may be generated when this property value is set.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.SecureCipher Property

Gets a value that specifies the encryption algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureCipher As SecureCipherAlgorithm
```

[C#]

```
public InternetMail.SecureCipherAlgorithm SecureCipher {get;}
```

Property Value

A SecureCipherAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureCipher** property returns a value which identifies the algorithm used to encrypt the data stream. If a secure connection has not been established, this property will return a value of **cipherNone**.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.SecureHash Property

Gets a value that specifies the message digest algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureHash As SecureHashAlgorithm
```

[C#]

```
public InternetMail.SecureHashAlgorithm SecureHash {get;}
```

Property Value

A SecureHashAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureHash** property returns a value which identifies the message digest (hash) algorithm that was selected when a secure connection was established. If a secure connection has not been established, this property will return a value of **hashNone**.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.SecureKeyExchange Property

Gets a value that specifies the key exchange algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureKeyExchange As SecureKeyAlgorithm
```

[C#]

```
public InternetMail.SecureKeyAlgorithm SecureKeyExchange {get;}
```

Property Value

A SecureKeyAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureKeyExchange** property returns a value which identifies the key exchange algorithm that was selected when a secure connection was established. If a secure connection has not been established, this property will return a value of **keyExchangeNone**.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.SecureProtocol Property

Gets and sets a value which specifies the protocol used for a secure connection.

[Visual Basic]

```
Public Property SecureProtocol As SecurityProtocols
```

[C#]

```
public InternetMail.SecurityProtocols SecureProtocol {get; set;}
```

Property Value

A SecurityProtocols enumeration value which identifies the protocol to be used when establishing a secure connection.

Remarks

The **SecureProtocol** property can be used to specify the security protocol to be used when establishing a secure connection with a server or accepting a secure connection from a client. By default, the class will attempt to use either SSL v3 or TLS v1 to establish the connection, with the appropriate protocol automatically selected based on the capabilities of the remote host. It is recommended that you only change this property value if you fully understand the implications of doing so. Assigning a value to this property will override the default protocol and force the class to attempt to use only the protocol specified.

Multiple security protocols may be specified by combining them using a bitwise **or** operator. After a connection has been established, this property will identify the protocol that was selected. Attempting to set this property after a connection has been established will result in an exception being thrown. This property should only be set after setting the **Secure** property to **true** and before calling the **Accept** or **Connect** methods.

In some cases, a server may only accept a secure connection if the TLS v1 protocol is specified. If the security protocol is not compatible with the server, then the connection will fail with an error indicating that the control is unable to establish a security context for the session. In this case, try assigning the property to **protocolTLS1** and attempt the connection again.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Sender Property

Gets and sets the address of the user who originated the message.

[Visual Basic]

```
Public Property Sender As String
```

[C#]

```
public string Sender {get; set;}
```

Property Value

A string which specifies the sender's email address.

Remarks

The **Sender** property returns the address of the user who originated the message. Setting this property updates the X-Sender header with the specified value.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.ServerName Property

Gets and sets a value which specifies the host name or address of the mail server.

[Visual Basic]

```
Public Property ServerName As String
```

[C#]

```
public string ServerName {get; set;}
```

Property Value

A string which specifies a server host name or address.

Remarks

The **ServerName** property can be used to set the host name for the mail server that you want to establish a connection to.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.ServerPort Property

Gets and sets a value which specifies the remote port number.

[Visual Basic]

```
Public Property ServerPort As Integer
```

[C#]

```
public int ServerPort {get; set;}
```

Property Value

An integer value which specifies a port number.

Remarks

The **ServerPort** property is used to set the port number that will be used to establish a connection with a mail server.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.ServerType Property

Gets and sets a value which specifies the type of mail server the client is connecting to.

[Visual Basic]
```
Public Property ServerType As MailServerType
```

[C#]
```
public InternetMail.MailServerType ServerType {get; set;}
```

Property Value

Returns a MailServerType enumeration value which specifies the type of mail server, either using the Post Office Protocol or the Internet Message Access Protocol.

Remarks

If this property value is specified as **serverUnknown**, the actual server type will be determined by the value of the **ServerPort** property, if the port number is recognized as a standard service port. If the server type cannot be automatically identified, an error will be returned when the **Connect** method is called.

If you are connecting using a non-standard port number, you should always initialize this property value to the correct server type before attempting to establish a connection.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Subject Property

Gets and sets the subject of the current message.

[Visual Basic]

```
Public Property Subject As String
```

[C#]

```
public string Subject {get; set;}
```

Property Value

A string which specifies the subject of the message.

Remarks

The **Subject** property returns the subject of the current message. Setting this property updates the Subject header with the specified value. Note that not all messages have subjects, in which case this property will be set to an empty string.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Subscribed Property

Gets a value that specifies if the user has subscribed to the currently selected mailbox.

[Visual Basic]

```
Public Property Subscribed As Boolean
```

[C#]

```
public bool Subscribed {get; set;}
```

Property Value

A boolean value that specifies if the user has subscribed to the current mailbox.

Remarks

The **Subscribed** property is used to determine if the current mailbox has been subscribed to by the user. If the property returns **false**, the server has indicated that the user has not subscribed to the mailbox. If the property returns **true**, the current mailbox is in the user's subscription list.

Setting the **Subscribed** property changes the subscription status of the current mailbox. Setting the property to **true** adds the mailbox to the user's list of subscribed mailboxes, while setting it to **false** removes the mailbox from the subscription list.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]

```
Public Property ThrowError As Boolean
```

[C#]

```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Timeout Property

Gets and sets a value which specifies a timeout period in seconds.

[Visual Basic]

```
Public Property Timeout As Integer
```

[C#]

```
public int Timeout {get; set;}
```

Property Value

An integer value which specifies a timeout period in seconds.

Remarks

Setting the **Timeout** property specifies the number of seconds until a blocking operation fails and returns an error.

For most applications it is recommended the timeout period be set between 10 and 20 seconds.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.TimeZone Property

Gets and sets the current timezone offset in seconds.

[Visual Basic]

```
Public Property TimeZone As Integer
```

[C#]

```
public int TimeZone {get; set;}
```

Property Value

An integer value which specifies the current timezone offset.

Remarks

The **TimeZone** property returns the current offset from UTC in seconds. Setting the property changes the current timezone offset to the specified value. The value of this property is initially determined by the date and time settings on the local system.

The **TimeZone** property value is used in conjunction with the **Localize** property to control how message date and time localization is handled.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.To Property

Gets and sets the address of the message recipient.

[Visual Basic]

```
Public Property To As String
```

[C#]

```
public string To {get; set;}
```

Property Value

A string which specifies the recipient of the message.

Remarks

The **To** property returns the address of the message recipient. Setting this property causes the To header to be updated with the specified value. Multiple addresses can be specified by separating them with commas.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Trace Property

Gets and sets a value which indicates if network function tracing is enabled.

[Visual Basic]
```
Public Property Trace As Boolean
```

[C#]
```
public bool Trace {get; set;}
```

Property Value

Returns **true** if network function tracing is enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Trace** property is used to enable (or disable) the tracing of network function calls. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the SocketTools networking classes and is enabled or disabled for an entire process. This means that once trace logging is enabled for a given component, all of the function calls made by the process using any of the SocketTools classes will be logged. For example, if you have an application using both the File Transfer Protocol and Post Office Protocol classes, and you set the **Trace** property to **true**, function calls made by both classes will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all classes used by the process.

If trace logging is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the logfile is fairly large. Since logfiles can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

When redistributing your application, make sure that you include the **SocketTools10.TraceLog.dll** module with your installation. If this library is not present, then no trace output will be generated and the value of the **Trace** property will be ignored. Only those function calls made by the SocketTools networking classes will be logged. Calls made directly to the Windows Sockets API, or calls made by other classes, will not be logged.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.TraceFile Property

Gets and sets a value which specifies the name of the client function tracing logfile.

[Visual Basic]

```
Public Property TraceFile As String
```

[C#]

```
public string TraceFile {get; set;}
```

Property Value

A string which specifies the name of the file.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named **SocketTools.log** is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFile** property should be set to the same value for each instance. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
 MyApp INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0 MyApp WRN:
connect(46, 192.0.0.1:1234, 16) returned -1 [10035] MyApp ERR: accept(46,
NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the control is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Note that if the specified file cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.TraceFlags Property

Gets and sets a value which specifies the client function tracing flags.

[Visual Basic]

```
Public Property TraceFlags As TraceOptions
```

[C#]

```
public InternetMail.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration which specifies the amount of detail written to the trace logfile.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Because network function tracing is enabled per-process, the trace flags are shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFlags** property should be set to the same value for each instance. Changing the trace flags for any one instance of the class will affect the logging performed for all SocketTools classes used by the application.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being written and the error **errorOperationWouldBlock** occurs, a warning is generated because the application simply needs to attempt to write the data at a later time.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.UserName Property

Gets and sets the username used to authenticate the client session.

[Visual Basic]

```
Public Property UserName As String
```

[C#]

```
public string UserName {get; set;}
```

Property Value

A string which specifies the username.

Remarks

If a username is not specified when the **Connect** method is called, the value of this property will be used as the default username when establishing a connection with the server.

See Also

InternetMail Class | SocketTools Namespace | BearerToken Property | Password Property | Connect Method

InternetMail.Version Property

Gets a value which returns the current version of the InternetMail class library.

[Visual Basic]

```
Public ReadOnly Property Version As String
```

[C#]

```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the InternetMail class library. This value can be used by an application for validation and debugging purposes.

See Also

InternetMail Class | SocketTools Namespace

InternetMail Methods

The methods of the **InternetMail** class are listed below. For a complete list of **InternetMail** class members, see the InternetMail Members topic.

Public Instance Methods

AppendMessage	Append text to the body of the current message part.
AttachData	Overloaded. Attach the contents of a byte array to the current message.
AttachFile	Overloaded. Attach the specified file to the current message.
AttachImage	Overloaded. Attach an inline image to the current message.
AttachThread	Obsolete. Attach an instance of the class to the current thread.
Cancel	Cancel the current blocking client operation.
ChangePassword	Change the mailbox password for the current user.
CheckMailbox	Create a checkpoint for the currently selected mailbox.
ClearMessage	Clear the header and body of the current message.
ComposeMessage	Overloaded. Compose a new mail message.
Connect	Overloaded. Establish a connection with a mail server.
CopyMessage	Copy a message from the current mailbox to another mailbox.
CreateMailbox	Creates a new mailbox on the server.
CreateMessage	Overloaded. Create a new message.
CreatePart	Overloaded. Create a new message part in a multipart message.
DeleteHeader	Overloaded. Delete a header field from the specified message part.
DeleteMailbox	Overloaded. Deletes a mailbox from the server.
DeleteMessage	Flags a message for deletion from the current mailbox.
DeletePart	Delete the specified message part from the current message.
Disconnect	Terminate the connection with the remote server.
Dispose	Overloaded. Releases all resources used by InternetMail.

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
ExportMessage	Overloaded. Export the current message to a file on the local system.
ExtractAllFiles	Overloaded. Extract all file attachments from the current message.
ExtractFile	Overloaded. Extract the contents of a file attachment and store it on the local system.
FindAttachment	Search for a specific file attachment in the current message.
GetFirstHeader	Return the first header in the current message part.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetHeader	Overloaded. Return the value of a header field in the specified message part.
GetHeaders	Overloaded. Retrieves the headers for the specified message from the server.
GetMessage	Overloaded. Retrieve a message from the server and return the contents in a byte array.
GetNextHeader	Return the next header in the current message part.
GetType (inherited from Object)	Gets the Type of the current instance.
Idle	Overloaded. Enables mailbox status monitoring for the client session.
ImportMessage	Replace the current message with the contents of a file.
Initialize	Overloaded. Initialize an instance of the InternetMail class.
ParseAddress	Overloaded. Parse an Internet email address.
ParseMessage	Parse the specified string, adding the contents to the current message.
RenameMailbox	Change the name of a mailbox.
ReselectMailbox	Reselects the current mailbox.
Reset	Reset the internal state of the object, resetting all properties to their default values.
SearchMailbox	Overloaded. Search the current mailbox for messages that match the specified criteria and character set.
SelectMailbox	Selects the specified mailbox for read-write access.
SendMessage	Overloaded. Submit the specified message to a

	mail server for delivery.
SetHeader	Overloaded. Set the value for a header in the specified message part.
StoreMessage	Overloaded. Retrieve a message from the current mailbox and store it in a file on the local system.
SubscribeMailbox	Overloaded. Subscribes the user to the specified mailbox.
ToString (inherited from Object)	Returns a String that represents the current Object.
UndeleteMessage	Removes the deletion flag for the specified message.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
UnselectMailbox	Overloaded. Unselects the current mailbox.
UnsubscribeMailbox	Overloaded. Unsubscribes the user from the specified mailbox.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the InternetMail class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the current message.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.AppendMessage Method

Append text to the body of the current message part.

[Visual Basic]

```
Public Function AppendMessage( _
   ByVal messageText As String _
) As Boolean
```

[C#]

```
public bool AppendMessage(
   string messageText
);
```

Parameters

messageText

A string which specifies the message text to be appended to the current message part.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.AttachData Method

Attach the contents of a byte array to the current message.

Overload List

Attach the contents of a byte array to the current message.

public bool AttachData(byte[],int);

Attach the contents of a byte array to the current message.

public bool AttachData(byte[],int,string);

Attach the contents of a byte array to the current message.

public bool AttachData(byte[],int,string,string);

Attach the contents of a byte array to the current message.

public bool AttachData(byte[],int,string,string,MimeAttachment);

Attach the contents of a string to the current message.

public bool AttachData(string);

Attach the contents of a string to the current message.

public bool AttachData(string,string);

Attach the contents of a string to the current message.

public bool AttachData(string,string,string);

Attach the contents of a string to the current message.

public bool AttachData(string,string,string,MimeAttachment);

See Also

InternetMail Class | SocketTools Namespace

InternetMail.AttachData Method (Byte[], Int32)

Attach the contents of a byte array to the current message.

```
[Visual Basic]
Overloads Public Function AttachData( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Boolean
```

```
[C#]
public bool AttachData(
   byte[] buffer,
   int length
);
```

Parameters

buffer
A byte array that contains the data to be attached to the message.

length
An integer value which specifies the maximum number of bytes top copy from the buffer. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The contents of the buffer will be attached to the message as inline content.

The buffer will be examined to determine what kind of data it contains. If there is only text characters, then the content type will be specified as text/plain. If the buffer contains binary data, then the content type will be specified as application/octet-stream, which is appropriate for any type of data.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.AttachData Overload List

InternetMail.AttachData Method (Byte[], Int32, String)

Attach the contents of a byte array to the current message.

[Visual Basic]

```
Overloads Public Function AttachData( _
   ByVal buffer As Byte(), _
   ByVal length As Integer, _
   ByVal contentName As String _
) As Boolean
```

[C#]

```
public bool AttachData(
   byte[] buffer,
   int length,
   string contentName
);
```

Parameters

buffer

A byte array that contains the data to be attached to the message.

length

An integer value which specifies the maximum number of bytes top copy from the buffer. This value cannot be larger than the size of the buffer specified by the caller.

contentName

An string argument which specifies a name for the data being attached to the message. This typically is used as a file name by the mail client to store the data in. If this parameter is omitted or passed as an empty string then no name is defined and the data is attached as inline content. Note that if a file name is specified with a path, only the base file name will be used.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The buffer will be examined to determine what kind of data it contains. If there is only text characters, then the content type will be specified as text/plain. If the buffer contains binary data, then the content type will be specified as application/octet-stream, which is appropriate for any type of data.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.AttachData Overload List

InternetMail.AttachData Method (Byte[], Int32, String, String)

Attach the contents of a byte array to the current message.

[Visual Basic]

```
Overloads Public Function AttachData( _
   ByVal buffer As Byte(), _
   ByVal length As Integer, _
   ByVal contentName As String, _
   ByVal contentType As String _
) As Boolean
```

[C#]

```
public bool AttachData(
   byte[] buffer,
   int length,
   string contentName,
   string contentType
);
```

Parameters

buffer
A byte array that contains the data to be attached to the message.

length
An integer value which specifies the maximum number of bytes top copy from the buffer. This value cannot be larger than the size of the buffer specified by the caller.

contentName
An string argument which specifies a name for the data being attached to the message. This typically is used as a file name by the mail client to store the data in. Note that if a file name is specified with a path, only the base file name will be used.

contentType
An string argument which specifies the type of data being attached. The value must be a valid MIME content type. If the buffer contains only text characters, then the content type will be specified as text/plain. If the buffer contains binary data, then the content type will be specified as application/octet-stream, which is appropriate for any type of data.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.AttachData Overload List

InternetMail.AttachData Method (Byte[], Int32, String, String, MimeAttachment)

Attach the contents of a byte array to the current message.

[Visual Basic]

```
Overloads Public Function AttachData( _
   ByVal buffer As Byte(), _
   ByVal length As Integer, _
   ByVal contentName As String, _
   ByVal contentType As String, _
   ByVal options As MimeAttachment _
) As Boolean
```

[C#]

```
public bool AttachData(
   byte[] buffer,
   int length,
   string contentName,
   string contentType,
   MimeAttachment options
);
```

Parameters

buffer

A byte array that contains the data to be attached to the message.

length

An integer value which specifies the maximum number of bytes top copy from the buffer. This value cannot be larger than the size of the buffer specified by the caller.

contentName

An string argument which specifies a name for the data being attached to the message. This typically is used as a file name by the mail client to store the data in. Note that if a file name is specified with a path, only the base file name will be used.

contentType

An string argument which specifies the type of data being attached. The value must be a valid MIME content type. If the buffer contains only text characters, then the content type will be specified as text/plain. If the buffer contains binary data, then the content type will be specified as application/octet-stream, which is appropriate for any type of data.

options

A MimeAttachment enumeration which specifies how the data should be attached to the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.AttachData Overload List

InternetMail.AttachData Method (String)

Attach the contents of a string to the current message.

[Visual Basic]

```
Overloads Public Function AttachData( _
   ByVal buffer As String _
) As Boolean
```

[C#]

```
public bool AttachData(
   string buffer
);
```

Parameters

buffer

A string that contains the data to be attached to the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The contents of the buffer will be attached to the message as inline content with a content type of text/plain.

This implementation of the method should never be used to attach binary data to a message. If you need to attach binary data, use the implementation of that accepts a byte array as the ***buffer*** parameter.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.AttachData Overload List

InternetMail.AttachData Method (String, String)

Attach the contents of a string to the current message.

[Visual Basic]

```
Overloads Public Function AttachData( _
   ByVal buffer As String, _
   ByVal contentName As String _
) As Boolean
```

[C#]

```
public bool AttachData(
   string buffer,
   string contentName
);
```

Parameters

buffer

A string that contains the data to be attached to the message.

contentName

An string argument which specifies a name for the data being attached to the message. This typically is used as a file name by the mail client to store the data in. Note that if a file name is specified with a path, only the base file name will be used.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The content will be attached using the content type of text/plain.

This implementation of the method should never be used to attach binary data to a message. If you need to attach binary data, use the implementation of that accepts a byte array as the ***buffer*** parameter.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.AttachData Overload List

InternetMail.AttachData Method (String, String, String)

Attach the contents of a string to the current message.

[Visual Basic]

```
Overloads Public Function AttachData( _
   ByVal buffer As String, _
   ByVal contentName As String, _
   ByVal contentType As String _
) As Boolean
```

[C#]

```
public bool AttachData(
   string buffer,
   string contentName,
   string contentType
);
```

Parameters

buffer

A string that contains the data to be attached to the message.

contentName

An string argument which specifies a name for the data being attached to the message. This typically is used as a file name by the mail client to store the data in.Note that if a file name is specified with a path, only the base file name will be used.

contentType

An string argument which specifies the type of data being attached. The value must be a valid MIME content type. If the buffer contains only text characters, then the content type will be specified as "text/plain". If the buffer contains binary data, then the content type will be specified as "application/octet-stream", which is appropriate for any type of data.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This implementation of the method should never be used to attach binary data to a message. If you need to attach binary data, use the implementation of that accepts a byte array as the ***buffer*** parameter.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.AttachData Overload List

InternetMail.AttachData Method (String, String, String, MimeAttachment)

Attach the contents of a string to the current message.

[Visual Basic]

```
Overloads Public Function AttachData( _
   ByVal buffer As String, _
   ByVal contentName As String, _
   ByVal contentType As String, _
   ByVal options As MimeAttachment _
) As Boolean
```

[C#]

```
public bool AttachData(
   string buffer,
   string contentName,
   string contentType,
   MimeAttachment options
);
```

Parameters

buffer
A string that contains the data to be attached to the message.

contentName
An string argument which specifies a name for the data being attached to the message. This typically is used as a file name by the mail client to store the data in.Note that if a file name is specified with a path, only the base file name will be used.

contentType
An string argument which specifies the type of data being attached. The value must be a valid MIME content type. If the buffer contains only text characters, then the content type will be specified as text/plain. If the buffer contains binary data, then the content type will be specified as application/octet-stream, which is appropriate for any type of data.

options
A MimeAttachment enumeration which specifies how the data should be attached to the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This implementation of the method should never be used to attach binary data to a message. If you need to attach binary data, use the implementation of that accepts a byte array as the *buffer* parameter.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.AttachData Overload List

InternetMail.AttachFile Method

Attach the specified file to the current message.

Overload List

Attach the specified file to the current message.

public bool AttachFile(string);

Attach the specified file to the current message.

public bool AttachFile(string,MimeAttachment);

See Also

InternetMail Class | SocketTools Namespace

InternetMail.AttachFile Method (String)

Attach the specified file to the current message.

[Visual Basic]

```
Overloads Public Function AttachFile( _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool AttachFile(
   string fileName
);
```

Parameters

fileName

A string which specifies the name of the file to be attached to the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AttachFile** method attaches the specified file to the current message. If the message already contains one or more file attachments, then it is added to the end of the message. If the message does not contain any attached files, then it is converted to a multipart message and the file is appended to the message.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.AttachFile Overload List

InternetMail.AttachFile Method (String, MimeAttachment)

Attach the specified file to the current message.

[Visual Basic]

```
Overloads Public Function AttachFile( _
   ByVal fileName As String, _
   ByVal options As MimeAttachment _
) As Boolean
```

[C#]

```
public bool AttachFile(
   string fileName,
   MimeAttachment options
);
```

Parameters

fileName
A string which specifies the name of the file to be attached to the message.

options
A MimeAttachment enumeration which specifies how the data should be attached to the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AttachFile** method attaches the specified file to the current message. If the message already contains one or more file attachments, then it is added to the end of the message. If the message does not contain any attached files, then it is converted to a multipart message and the file is appended to the message.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.AttachFile Overload List

InternetMail.AttachThread Method

NOTE: This method is now obsolete.

The AttachThread method has been deprecated

Attach an instance of the class to the current thread.

[Visual Basic]

```
<Obsolete(Message:="The AttachThread method has been deprecated", IsError:=False)>
Public Function AttachThread() As Boolean
```

[C#]

```
[Obsolete(Message="The AttachThread method has been deprecated", IsError=False)]
public bool AttachThread();
```

Return Value

A boolean value which specifies if the message could be attached to the current thread. If this method returns **false**, the message could not be attached to the thread and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method has been deprecated and should no longer be used. The current version of the InternetMail class uses a free threading model which permits any thread to access methods and properties. However, applications must take care to synchronize access to the class instance across multiple threads if they are modifying the message contents.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Cancel Method

Cancel the current blocking client operation.

[Visual Basic]

```
Public Sub Cancel()
```

[C#]

```
public void Cancel();
```

Remarks

When the **Cancel** method is called, the blocking client operation will not immediately fail. An internal flag is set which causes the blocking operation to exit with an error. This means that the application cannot cancel an operation and immediately perform some other blocking function. Instead it must allow the calling stack to unwind, returning back to the blocking operation before making any further function calls.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.ChangePassword Method

Change the mailbox password for the current user.

[Visual Basic]

```
Public Function ChangePassword( _
   ByVal userName As String, _
   ByVal oldPassword As String, _
   ByVal newPassword As String _
) As Boolean
```

[C#]

```
public bool ChangePassword(
   string userName,
   string oldPassword,
   string newPassword
);
```

Parameters

userName
A string which specifies the username for which the password will be changed.

oldPassword
A string which specifies the current password.

newPassword
A string which specifies the new password.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ChangePassword** method changes the current password that will be used to authenticate the user. Once the password has been changed, the **Password** property will be updated with the new password.

Note that in order to change the user's mailbox password, the server must be running the poppass service on port 106, on the same server. Because passwords are transmitted as clear text (unencrypted), this service is not considered secure and may not be available.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.CheckMailbox Method

Create a checkpoint for the currently selected mailbox.

[Visual Basic]
```
Public Function CheckMailbox() As Boolean
```

[C#]
```
public bool CheckMailbox();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CheckMailbox** method requests that the server create a checkpoint of the currently selected mailbox, and updates the current number of new, unread messages available to the client.

When the client requests a checkpoint, the server may perform implementation-dependent housekeeping for that mailbox, such updating the mailbox on disk with the current state of the mailbox in memory. On some systems this command has no effect other than to update the client with the current number of messages in the mailbox.

This function actually sends two IMAP commands. The first is the CHECK command, followed by the NOOP command to poll for any new messages that have arrived. In addition to polling the server for new messages, this command can also be used to ensure the idle timer on the server does not expire and force a disconnect from the client.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.ClearMessage Method

Clear the header and body of the current message.

[Visual Basic]

```
Public Function ClearMessage() As Boolean
```

[C#]

```
public bool ClearMessage();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ClearMessage** method clears the current message, releasing the memory allocated for the message and any attachments.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.ComposeMessage Method

Compose a new mail message.

Overload List

Compose a new mail message.

public bool ComposeMessage(string,string,string,string);

Compose a new mail message.

public bool ComposeMessage(string,string,string,string,string);

Compose a new mail message.

public bool ComposeMessage(string,string,string,string,string,string);

Compose a new mail message.

public bool ComposeMessage(string,string,string,string,string,string,string);

Compose a new mail message.

public bool ComposeMessage(string,string,string,string,string,string,string,MimeCharacterSet,MimeEncoding);

See Also

InternetMail Class | SocketTools Namespace

InternetMail.ComposeMessage Method (String, String, String, String)

Compose a new mail message.

[Visual Basic]

```
Overloads Public Function ComposeMessage( _
   ByVal messageFrom As String, _
   ByVal messageTo As String, _
   ByVal messageSubject As String, _
   ByVal messageText As String _
) As Boolean
```

[C#]

```
public bool ComposeMessage(
   string messageFrom,
   string messageTo,
   string messageSubject,
   string messageText
);
```

Parameters

messageFrom

A string argument which specifies the sender's email address. Only a single address should be specified. After the message has been composed, the **From** property will be updated with this value.

messageTo

A string argument which specifies one or more recipient email addresses. Multiple email addresses may be specified by separating them with commas. After the message has been composed, the **To** property will be updated with this value.

messageSubject

A string argument which specifies the subject for the message. If the argument is not specified, then no Subject header field will be created for this message. After the message has been composed, the **Subject** property will be updated with this value

messageText

An string argument which specifies the body of the message. Each line of text contained in the string should be terminated with a carriage-return/linefeed (CRLF) pair, which is recognized as the end-of-line.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ComposeMessage** method creates a new mail message, or replaces the current message if one already exists.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.ComposeMessage Overload List

InternetMail.ComposeMessage Method (String, String, String, String, String)

Compose a new mail message.

[Visual Basic]

```
Overloads Public Function ComposeMessage( _
   ByVal messageFrom As String, _
   ByVal messageTo As String, _
   ByVal messageCc As String, _
   ByVal messageSubject As String, _
   ByVal messageText As String _
) As Boolean
```

[C#]

```
public bool ComposeMessage(
   string messageFrom,
   string messageTo,
   string messageCc,
   string messageSubject,
   string messageText
);
```

Parameters

messageFrom
A string argument which specifies the sender's email address. Only a single address should be specified. After the message has been composed, the **From** property will be updated with this value.

messageTo
A string argument which specifies one or more recipient email addresses. Multiple email addresses may be specified by separating them with commas. After the message has been composed, the **To** property will be updated with this value.

messageCc
A string argument which specifies one or more additional recipient addresses that will receive a copy of the message. If this argument is not specified, then no Cc header field will be created for this message. After the message has been composed, the **Cc** property will be updated with this value.

messageSubject
A string argument which specifies the subject for the message. If the argument is not specified, then no Subject header field will be created for this message. After the message has been composed, the **Subject** property will be updated with this value

messageText
An string argument which specifies the body of the message. Each line of text contained in the string should be terminated with a carriage-return/linefeed (CRLF) pair, which is recognized as the end-of-line.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ComposeMessage** method creates a new mail message, or replaces the current message if one already exists.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.ComposeMessage Overload List

InternetMail.ComposeMessage Method (String, String, String, String, String, String)

Compose a new mail message.

[Visual Basic]

```
Overloads Public Function ComposeMessage( _
   ByVal messageFrom As String, _
   ByVal messageTo As String, _
   ByVal messageCc As String, _
   ByVal messageBcc As String, _
   ByVal messageSubject As String, _
   ByVal messageText As String _
) As Boolean
```

[C#]

```
public bool ComposeMessage(
   string messageFrom,
   string messageTo,
   string messageCc,
   string messageBcc,
   string messageSubject,
   string messageText
);
```

Parameters

messageFrom
: A string argument which specifies the sender's email address. Only a single address should be specified. After the message has been composed, the **From** property will be updated with this value.

messageTo
: A string argument which specifies one or more recipient email addresses. Multiple email addresses may be specified by separating them with commas. After the message has been composed, the **To** property will be updated with this value.

messageCc
: A string argument which specifies one or more additional recipient addresses that will receive a copy of the message. If this argument is not specified, then no Cc header field will be created for this message. After the message has been composed, the **Cc** property will be updated with this value.

messageBcc
: A string argument which specifies one or more additional recipient addresses that will receive a "blind" copy of the message. If this argument is not specified, then no Bcc header field will be created for this message. After the message has been composed, the **Bcc** property will be updated with this value. Note that the Bcc header field is not normally included in the header when the message is exported.

messageSubject
: A string argument which specifies the subject for the message. If the argument is not specified, then no Subject header field will be created for this message. After the message has been composed, the **Subject** property will be updated with this value

messageText
: An string argument which specifies the body of the message. Each line of text contained in the string should be terminated with a carriage-return/linefeed (CRLF) pair, which is recognized as the end-of-line.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ComposeMessage** method creates a new mail message, or replaces the current message if one already exists.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.ComposeMessage Overload List

InternetMail.ComposeMessage Method (String, String, String, String, String, String, String)

Compose a new mail message.

[Visual Basic]

```
Overloads Public Function ComposeMessage( _
   ByVal messageFrom As String, _
   ByVal messageTo As String, _
   ByVal messageCc As String, _
   ByVal messageBcc As String, _
   ByVal messageSubject As String, _
   ByVal messageText As String, _
   ByVal messageHTML As String _
) As Boolean
```

[C#]

```
public bool ComposeMessage(
   string messageFrom,
   string messageTo,
   string messageCc,
   string messageBcc,
   string messageSubject,
   string messageText,
   string messageHTML
);
```

Parameters

messageFrom

A string argument which specifies the sender's email address. Only a single address should be specified. After the message has been composed, the **From** property will be updated with this value.

messageTo

A string argument which specifies one or more recipient email addresses. Multiple email addresses may be specified by separating them with commas. After the message has been composed, the **To** property will be updated with this value.

messageCc

A string argument which specifies one or more additional recipient addresses that will receive a copy of the message. If this argument is not specified, then no Cc header field will be created for this message. After the message has been composed, the **Cc** property will be updated with this value.

messageBcc

A string argument which specifies one or more additional recipient addresses that will receive a "blind" copy of the message. If this argument is not specified, then no Bcc header field will be created for this message. After the message has been composed, the **Bcc** property will be updated with this value. Note that the Bcc header field is not normally included in the header when the message is exported.

messageSubject

A string argument which specifies the subject for the message. If the argument is not specified, then no Subject header field will be created for this message. After the message has been composed, the **Subject** property will be updated with this value

messageText

An string argument which specifies the body of the message. Each line of text contained in the string should be terminated with a carriage-return/linefeed (CRLF) pair, which is recognized as the end-of-line.

messageHTML

A string argument which specifies an alternate HTML formatted message. If the ***messageText*** argument has been specified, then a multipart message will be created with both plain text and HTML text as the alternative. This allows mail clients to select which message body they wish to display. If the ***messageText*** argument is an empty string, then the message will only contain HTML. Although this is supported, it is not recommended because older mail clients may be unable to display the message correctly.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ComposeMessage** method creates a new mail message, or replaces the current message if one already exists.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.ComposeMessage Overload List

InternetMail.ComposeMessage Method (String, String, String, String, String, String, String, MimeCharacterSet, MimeEncoding)

Compose a new mail message.

[Visual Basic]

```
Overloads Public Function ComposeMessage( _
   ByVal messageFrom As String, _
   ByVal messageTo As String, _
   ByVal messageCc As String, _
   ByVal messageBcc As String, _
   ByVal messageSubject As String, _
   ByVal messageText As String, _
   ByVal messageHTML As String, _
   ByVal characterSet As MimeCharacterSet, _
   ByVal encodingType As MimeEncoding _
) As Boolean
```

[C#]

```
public bool ComposeMessage(
   string messageFrom,
   string messageTo,
   string messageCc,
   string messageBcc,
   string messageSubject,
   string messageText,
   string messageHTML,
   MimeCharacterSet characterSet,
   MimeEncoding encodingType
);
```

Parameters

messageFrom

A string argument which specifies the sender's email address. Only a single address should be specified. After the message has been composed, the **From** property will be updated with this value.

messageTo

A string argument which specifies one or more recipient email addresses. Multiple email addresses may be specified by separating them with commas. After the message has been composed, the **To** property will be updated with this value.

messageCc

A string argument which specifies one or more additional recipient addresses that will receive a copy of the message. If this argument is not specified, then no Cc header field will be created for this message. After the message has been composed, the **Cc** property will be updated with this value.

messageBcc

A string argument which specifies one or more additional recipient addresses that will receive a "blind" copy of the message. If this argument is not specified, then no Bcc header field will be created for this message. After the message has been composed, the **Bcc** property will be updated with this value. Note that the Bcc header field is not normally included in the header when the message is exported.

messageSubject

A string argument which specifies the subject for the message. If the argument is not specified, then no Subject header field will be created for this message. After the message has been composed, the **Subject** property will be updated with this value

messageText
An string argument which specifies the body of the message. Each line of text contained in the string should be terminated with a carriage-return/linefeed (CRLF) pair, which is recognized as the end-of-line.

messageHTML
A string argument which specifies an alternate HTML formatted message. If the ***messageText*** argument has been specified, then a multipart message will be created with both plain text and HTML text as the alternative. This allows mail clients to select which message body they wish to display. If the ***messageText*** argument is an empty string, then the message will only contain HTML. Although this is supported, it is not recommended because older mail clients may be unable to display the message correctly.

characterSet
A MimeCharacterSet enumeration value which specifies the character set to use when composing the message.

encodingType
A MimeEncoding enumeration value which specifies the encoding type to use when composing the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ComposeMessage** method creates a new mail message, or replaces the current message if one already exists.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.ComposeMessage Overload List

InternetMail.Connect Method

Establish a connection with a remote host.

Overload List

Establish a connection with a remote host.

public bool Connect();

Establish a connection with a remote host.

public bool Connect(string,int);

Establish a connection with a remote host.

public bool Connect(string,int,int);

Establish a connection with a remote host.

public bool Connect(string,int,int,InternetMailOptions);

Establish a connection with a remote host.

public bool Connect(string,int,string,string);

Establish a connection with a remote host.

public bool Connect(string,int,string,string,int);

Establish a connection with a mail server.

public bool Connect(string,int,string,string,int,InternetMailOptions);

Establish a connection with a remote host.

public bool Connect(string,string,string);

See Also

InternetMail Class | SocketTools Namespace | BearerToken Property | Options Property | Password Property | ServerName Property | ServerPort Property | Timeout Property | UserName Property

InternetMail.Connect Method ()

Establish a connection with a remote host.

[Visual Basic]
```
Overloads Public Function Connect() As Boolean
```

[C#]
```
public bool Connect();
```

Return Value

A boolean value which specifies if the connection has been established. A return value of **true** indicates that the connection has completed and the application may exchange data with the mail server. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection using assigned property values.

The value of the **ServerName** will be used to determine the host name or address to connect to.

The value of the **RemotePort** property will be used to determine the port number to connect to.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.Connect Overload List | BearerToken Property | Options Property | Password Property | ServerName Property | ServerPort Property | Timeout Property | UserName Property

InternetMail.Connect Method (String, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

Return Value

A boolean value which specifies if the connection has been established. A return value of **true** indicates that the connection has completed and the application may exchange data with the mail server. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.Connect Overload List | BearerToken Property | Options Property | Password Property | ServerName Property | ServerPort Property | Timeout Property | UserName Property

InternetMail.Connect Method (String, Int32, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   int timeout
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller.

Return Value

A boolean value which specifies if the connection has been established. A return value of **true** indicates that the connection has completed and the application may exchange data with the mail server. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.Connect Overload List | BearerToken Property | Options Property | Password Property | ServerName Property | ServerPort Property | Timeout Property | UserName Property

InternetMail.Connect Method (String, Int32, Int32, InternetMailOptions)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer, _
   ByVal options As InternetMailOptions _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   int timeout,
   InternetMailOptions options
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller.

options

One or more of the InternetMailOptions enumeration flags.

Return Value

A boolean value which specifies if the connection has been established. A return value of **true** indicates that the connection has completed and the application may exchange data with the mail server. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.Connect Overload List | BearerToken Property | Options Property | Password Property | ServerName Property | ServerPort Property | Timeout Property | UserName Property

InternetMail.Connect Method (String, Int32, String, String)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName

A string which specifies a username used to authenticate the client session.

userPassword

A string which specifies the password used to authenticate the client session. If the **BearerToken** property has been set, this parameter is ignored and OAuth 2.0 will be used for authentication.

Return Value

A boolean value which specifies if the connection has been established. A return value of **true** indicates that the connection has completed and the application may exchange data with the mail server. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.Connect Overload List | BearerToken Property | Options Property | Password Property | ServerName Property | ServerPort Property | Timeout Property | UserName Property

InternetMail.Connect Method (String, Int32, String, String, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   int timeout
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName

A string which specifies a username used to authenticate the client session.

userPassword

A string which specifies the password used to authenticate the client session. If the **BearerToken** property has been set, this parameter is ignored and OAuth 2.0 will be used for authentication.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller.

Return Value

A boolean value which specifies if the connection has been established. A return value of **true** indicates that the connection has completed and the application may exchange data with the mail server. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.Connect Overload List | BearerToken Property | Options Property | Password Property | ServerName Property | ServerPort Property | Timeout Property | UserName Property

InternetMail.Connect Method (String, Int32, String, String, Int32, InternetMailOptions)

Establish a connection with a mail server.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal timeout As Integer, _
   ByVal options As InternetMailOptions _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   int timeout,
   InternetMailOptions options
);
```

Parameters

hostName
: A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort
: An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName
: A string which specifies a username used to authenticate the client session.

userPassword
: A string which specifies the password used to authenticate the client session. If the **BearerToken** property has been set, this parameter is ignored and OAuth 2.0 will be used for authentication.

timeout
: An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller.

options
: One or more of the InternetMailOptions enumeration flags.

Return Value

A boolean value which specifies if the connection has been established. A return value of **true** indicates that the connection has completed and the application may exchange data with the mail server. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.Connect Overload List | BearerToken Property |

Options Property | Password Property | ServerName Property | ServerPort Property | Timeout Property | UserName Property

InternetMail.Connect Method (String, String, String)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   string userName,
   string userPassword
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

userName

A string which specifies a username used to authenticate the client session.

userPassword

A string which specifies the password used to authenticate the client session. If the **BearerToken** property has been set, this parameter is ignored and OAuth 2.0 will be used for authentication.

Return Value

A boolean value which specifies if the connection has been established. A return value of **true** indicates that the connection has completed and the application may exchange data with the mail server. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.Connect Overload List | BearerToken Property | Options Property | Password Property | ServerName Property | ServerPort Property | Timeout Property | UserName Property

InternetMail.CopyMessage Method

Copy a message from the current mailbox to another mailbox.

[Visual Basic]

```
Public Function CopyMessage( _
   ByVal messageId As Integer, _
   ByVal mailboxName As String _
) As Boolean
```

[C#]

```
public bool CopyMessage(
   int messageId,
   string mailboxName
);
```

Parameters

messageId

The message identifier which specifies which message will be copied to the mailbox. This value must be greater than zero and specify a valid message number.

mailboxName

A string which specifies the name of the mailbox that the message will be copied to. The mailbox must already exist, and the client must have the appropriate access rights to modify the mailbox.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CopyMessage** method copies a message from the current mailbox to the specified mailbox. The message is appended to the mailbox, and the message flags and internal date are preserved. If the mailbox does not exist, the method will fail. To create a new mailbox, use the **CreateMailbox** method. A message can be copied within the same mailbox, in which case the server may flag it as a new message.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.CreateMailbox Method

Creates a new mailbox on the server.

```
[Visual Basic]
Public Function CreateMailbox( _
   ByVal mailboxName As String _
) As Boolean
```

```
[C#]
public bool CreateMailbox(
   string mailboxName
);
```

Parameters

mailboxName

A string which specifies the name of the new mailbox to be created.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateMailbox** method creates a new mailbox on the server. If the mailbox name is suffixed with the server's hierarchy delimiter, this indicates to the server that the client intends to create mailbox names under the specified name in the hierarchy. If superior hierarchical names are specified in the mailbox name, then the server may automatically create them as needed. For example, if the mailbox name "Mail/Office/Projects" is specified and "Mail/Office" does not exist, it may be automatically created by the server.

The special mailbox name INBOX is reserved, and cannot be created. It is recommended that mailbox names only consist of printable ASCII characters, and the special characters "*" and "%" should be avoided.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.CreateMessage Method

Create a new message.

Overload List

Create a new message.

public bool CreateMessage(byte[],int);

Create a new message.

public bool CreateMessage(byte[],int,ImapFlags);

Create a new message.

public bool CreateMessage(string);

Create a new message.

public bool CreateMessage(string,ImapFlags);

Create a new message.

public bool CreateMessage(string,byte[],int,ImapFlags);

Create a new message.

public bool CreateMessage(string,string,ImapFlags);

See Also

InternetMail Class | SocketTools Namespace

InternetMail.CreateMessage Method (Byte[], Int32)

Create a new message.

[Visual Basic]

```
Overloads Public Function CreateMessage( _
   ByVal message As Byte(), _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool CreateMessage(
   byte[] message,
   int length
);
```

Parameters

message
A byte array that contains the message data.

length
An integer value which specifies the size of the message in bytes. This value cannot be larger than the size of the message buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateMessage** method creates a new message, appending it to the contents of the specified mailbox. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The message will be created in the mailbox specified by the **MailboxName** property.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.CreateMessage Overload List

InternetMail.CreateMessage Method (Byte[], Int32, ImapFlags)

Create a new message.

[Visual Basic]

```
Overloads Public Function CreateMessage( _
   ByVal message As Byte(), _
   ByVal length As Integer, _
   ByVal messageFlags As ImapFlags _
) As Boolean
```

[C#]

```
public bool CreateMessage(
   byte[] message,
   int length,
   ImapFlags messageFlags
);
```

Parameters

message

A byte array that contains the message data.

length

An integer value which specifies the size of the message in bytes. This value cannot be larger than the size of the message buffer specified by the caller.

messageFlags

An ImapFlags enumeration value which specifies one or more message flags. One or more of the following flags may be used:

Flag	Description
imapFlagNone	The message will be created with no flags set.
imapFlagAnswered	The message has been answered.
imapFlagDraft	The message is a draft copy.
imapFlagUrgent	The message is urgent.
imapFlagSeen	The message has already been read.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateMessage** method creates a new message, appending it to the contents of the specified mailbox. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The message will be created in the mailbox specified by the **MailboxName** property.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.CreateMessage Overload List

InternetMail.CreateMessage Method (String)

Create a new message.

[Visual Basic]

```
Overloads Public Function CreateMessage( _
   ByVal message As String _
) As Boolean
```

[C#]

```
public bool CreateMessage(
   string message
);
```

Parameters

message

A string that contains the message data.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateMessage** method creates a new message, appending it to the contents of the specified mailbox. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The message will be created in the mailbox specified by the **MailboxName** property.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.CreateMessage Overload List

InternetMail.CreateMessage Method (String, ImapFlags)

Create a new message.

[Visual Basic]

```
Overloads Public Function CreateMessage( _
   ByVal message As String, _
   ByVal messageFlags As ImapFlags _
) As Boolean
```

[C#]

```
public bool CreateMessage(
   string message,
   ImapFlags messageFlags
);
```

Parameters

message

A string that contains the message data.

messageFlags

An ImapFlags enumeration value which specifies one or more message flags. One or more of the following flags may be used:

Flag	Description
imapFlagNone	The message will be created with no flags set.
imapFlagAnswered	The message has been answered.
imapFlagDraft	The message is a draft copy.
imapFlagUrgent	The message is urgent.
imapFlagSeen	The message has already been read.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateMessage** method creates a new message, appending it to the contents of the specified mailbox. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The message will be created in the mailbox specified by the **MailboxName** property.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.CreateMessage Overload List

InternetMail.CreateMessage Method (String, Byte[], Int32, ImapFlags)

Create a new message.

[Visual Basic]

```
Overloads Public Function CreateMessage( _
   ByVal mailboxName As String, _
   ByVal message As Byte(), _
   ByVal length As Integer, _
   ByVal messageFlags As ImapFlags _
) As Boolean
```

[C#]

```
public bool CreateMessage(
   string mailboxName,
   byte[] message,
   int length,
   ImapFlags messageFlags
);
```

Parameters

mailboxName

A string which specifies the name of the mailbox the message will be created in.

message

A byte array that contains the message data.

length

An integer value which specifies the size of the message in bytes. This value cannot be larger than the size of the message buffer specified by the caller.

messageFlags

An ImapFlags enumeration value which specifies one or more message flags. One or more of the following flags may be used:

Flag	Description
imapFlagNone	The message will be created with no flags set.
imapFlagAnswered	The message has been answered.
imapFlagDraft	The message is a draft copy.
imapFlagUrgent	The message is urgent.
imapFlagSeen	The message has already been read.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateMessage** method creates a new message, appending it to the contents of the specified mailbox. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.CreateMessage Overload List

InternetMail.CreateMessage Method (String, String, ImapFlags)

Create a new message.

```
[Visual Basic]
Overloads Public Function CreateMessage( _
   ByVal mailboxName As String, _
   ByVal message As String, _
   ByVal messageFlags As ImapFlags _
) As Boolean
```

```
[C#]
public bool CreateMessage(
   string mailboxName,
   string message,
   ImapFlags messageFlags
);
```

Parameters

mailboxName

A string which specifies the name of the mailbox the message will be created in.

message

A string that contains the message data.

messageFlags

An ImapFlags enumeration value which specifies one or more message flags. One or more of the following flags may be used:

Flag	Description
imapFlagNone	The message will be created with no flags set.
imapFlagAnswered	The message has been answered.
imapFlagDraft	The message is a draft copy.
imapFlagUrgent	The message is urgent.
imapFlagSeen	The message has already been read.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateMessage** method creates a new message, appending it to the contents of the specified mailbox. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.CreateMessage Overload List

InternetMail.CreatePart Method

Create an empty message part in a multipart message.

Overload List

Create an empty message part in a multipart message.

public bool CreatePart();

Create a new message part in a multipart message.

public bool CreatePart(string);

Create a new message part in a multipart message.

public bool CreatePart(string,MimeCharacterSet,MimeEncoding);

See Also

InternetMail Class | SocketTools Namespace

InternetMail.CreatePart Method ()

Create an empty message part in a multipart message.

[Visual Basic]

```
Overloads Public Function CreatePart() As Boolean
```

[C#]

```
public bool CreatePart();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreatePart** method creates a new empty message part. If the current message is a simple RFC822 formatted message, then this method converts it to a MIME multipart message. The current message part will be set to the new part that was just created.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.CreatePart Overload List

InternetMail.CreatePart Method (String)

Create a new message part in a multipart message.

[Visual Basic]

```
Overloads Public Function CreatePart( _
   ByVal messageText As String _
) As Boolean
```

[C#]

```
public bool CreatePart(
   string messageText
);
```

Parameters

messageText

A string argument which specifies the body of the new message part. Each line of text contained in the string should be terminated with a carriage-return/linefeed (CRLF) pair, which is recognized as the end-of-line.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreatePart** method creates a new message part. If the current message is a simple RFC822 formatted message, then this method converts it to a MIME multipart message. The current message part will be set to the new part that was just created.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.CreatePart Overload List

InternetMail.CreatePart Method (String, MimeCharacterSet, MimeEncoding)

Create a new message part in a multipart message.

[Visual Basic]

```
Overloads Public Function CreatePart( _
   ByVal messageText As String, _
   ByVal characterSet As MimeCharacterSet, _
   ByVal encodingType As MimeEncoding _
) As Boolean
```

[C#]

```
public bool CreatePart(
   string messageText,
   MimeCharacterSet characterSet,
   MimeEncoding encodingType
);
```

Parameters

messageText
: A string argument which specifies the body of the new message part. Each line of text contained in the string should be terminated with a carriage-return/linefeed (CRLF) pair, which is recognized as the end-of-line.

characterSet
: A MimeCharacterSet enumeration value which specifies the character set to use when composing the message.

encodingType
: A MimeEncoding enumeration value which specifies the encoding type to use when composing the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreatePart** method creates a new message part. If the current message is a simple RFC822 formatted message, then this method converts it to a MIME multipart message. The current message part will be set to the new part that was just created.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.CreatePart Overload List

InternetMail.DeleteHeader Method

Delete a header field from the specified message part.

Overload List

Delete a header field from the specified message part.

public bool DeleteHeader(int,string);

Delete a header field from the current message part.

public bool DeleteHeader(string);

See Also

InternetMail Class | SocketTools Namespace

InternetMail.DeleteHeader Method (Int32, String)

Delete a header field from the specified message part.

[Visual Basic]

```
Overloads Public Function DeleteHeader( _
   ByVal messagePart As Integer, _
   ByVal headerName As String _
) As Boolean
```

[C#]

```
public bool DeleteHeader(
   int messagePart,
   string headerName
);
```

Parameters

messagePart
An integer which specifies the message part.

headerName
A string which specifies the header field to delete from the specified message part.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

A message part of zero specifies the main message part which contains the standard headers such as To, From and Subject. The number of message parts in the current message is returned by the **PartCount** property.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.DeleteHeader Overload List

InternetMail.DeleteHeader Method (String)

Delete a header field from the current message part.

[Visual Basic]

```
Overloads Public Function DeleteHeader( _
   ByVal headerName As String _
) As Boolean
```

[C#]

```
public bool DeleteHeader(
   string headerName
);
```

Parameters

headerName

A string which specifies the header field to delete from the current message part.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The current message part is returned by the **Part** property.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.DeleteHeader Overload List

InternetMail.DeleteMailbox Method

Deletes the currently selected mailbox from the server.

Overload List

Deletes the currently selected mailbox from the server.

public bool DeleteMailbox();

Deletes a mailbox from the server.

public bool DeleteMailbox(string);

See Also

InternetMail Class | SocketTools Namespace

InternetMail.DeleteMailbox Method ()

Deletes the currently selected mailbox from the server.

[Visual Basic]
```
Overloads Public Function DeleteMailbox() As Boolean
```

[C#]
```
public bool DeleteMailbox();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **DeleteMailbox** method deletes the currently selected mailbox from the server. The current mailbox will be automatically unselected and any messages marked for deletion will be expunged before the mailbox is removed. If the delete operation fails, the client will remain in an unselected state until either the **ExamineMailbox** or **SelectMailbox** method is called

A mailbox cannot be deleted if it contains inferior hierarchical names and has the **imapFlagNoSelect** attribute. On most systems this is the case when the mailbox name references a directory on the server, and that directory contains other subdirectories or mailboxes. To remove the current mailbox, you must first delete any child mailboxes that exist.

The special mailbox named INBOX cannot be deleted.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.DeleteMailbox Overload List

InternetMail.DeleteMailbox Method (String)

Deletes a mailbox from the server.

[Visual Basic]
```
Overloads Public Function DeleteMailbox( _
   ByVal mailboxName As String _
) As Boolean
```

[C#]
```
public bool DeleteMailbox(
   string mailboxName
);
```

Parameters

mailboxName
A string which specifies the name of the new mailbox to be deleted.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **DeleteMailbox** method deletes a mailbox from the server. A mailbox cannot be deleted if it contains inferior hierarchical names and has the **imapFlagNoSelect** attribute. On most systems this is the case when the mailbox name references a directory on the server, and that directory contains other subdirectories or mailboxes. To remove the mailbox, you must first delete any child mailboxes that exist.

If the mailbox that is deleted is the currently selected mailbox, it will be automatically unselected and any messages marked for deletion will be expunged before the mailbox is removed. If the delete operation fails, the client will remain in an unselected state until either the **ExamineMailbox** or **SelectMailbox** method is called.

The special mailbox named INBOX cannot be deleted.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.DeleteMailbox Overload List

InternetMail.DeleteMessage Method

Flags a message for deletion from the current mailbox.

[Visual Basic]

```
Public Function DeleteMessage( _
   ByVal messageId As Integer _
) As Boolean
```

[C#]

```
public bool DeleteMessage(
   int messageId
);
```

Parameters

messageId

Number of message to delete from the server. This value must be greater than zero. The first message in the mailbox is message number one.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method only flags the message for deletion. The message is not actually deleted until the client disconnects from the server, however the deleted message will no longer be accessible to the client. To prevent deleted messages from actually being removed from the mailbox, call the **Reset** method.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.DeletePart Method

Delete the specified message part from the current message.

[Visual Basic]

```
Public Function DeletePart( _
   ByVal messagePart As Integer _
) As Boolean
```

[C#]

```
public bool DeletePart(
   int messagePart
);
```

Parameters

messagePart

An integer which specifies the message part.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method cannot be used to delete part zero, which is the main body of the message. Instead use the **ClearMessage** method to clear the contents of the entire message.

The number of message parts in the current message is returned by the **PartCount** property.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Disconnect Method

Terminate the connection with the remote server.

[Visual Basic]

```
Public Sub Disconnect()
```

[C#]

```
public void Disconnect();
```

Remarks

The **Disconnect** method terminates the connection with the remote host and closes the handle allocated by the class. Note that the handle is not immediately released when the connection is terminated and will enter a wait state for two minutes. After the time wait period has elapsed, the handle will be released by the operating system. This is a normal safety mechanism to handle any packets that may arrive after the connection has been closed.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Dispose Method

Releases all resources used by InternetMail.

Overload List

Releases all resources used by InternetMail.

public void Dispose();

Releases the unmanaged resources allocated by the InternetMail class and optionally releases the managed resources.

protected virtual void Dispose(bool);

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Dispose Method ()

Releases all resources used by InternetMail.

[Visual Basic]

```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]

```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.Dispose Overload List

InternetMail.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the InternetMail class and optionally releases the managed resources.

[Visual Basic]

```
Overridable Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]

```
protected virtual void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **InternetMail** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.Dispose Overload List

InternetMail.ExportMessage Method

Export the current message to a file on the local system.

Overload List

Export the current message to a file on the local system.

public bool ExportMessage(string);

Export the current message to a file on the local system.

public bool ExportMessage(string,MimeExportOptions);

See Also

InternetMail Class | SocketTools Namespace

InternetMail.ExportMessage Method (String)

Export the current message to a file on the local system.

[Visual Basic]

```
Overloads Public Function ExportMessage( _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool ExportMessage(
   string fileName
);
```

Parameters

fileName

A string which specifies the name of the file that will contain the message. If the file does not exist, it will be created. If it does exist, it will be overwritten with the contents of the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ExportMessage** method writes the current message to a file. If the file does not exist, it will be created. If it does exist, it will be overwritten with the contents of the message.

The value of the **Options** property determines the default export options, if any have been specified.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.ExportMessage Overload List

InternetMail.ExportMessage Method (String, MimeExportOptions)

Export the current message to a file on the local system.

[Visual Basic]

```
Overloads Public Function ExportMessage( _
   ByVal fileName As String, _
   ByVal options As MimeExportOptions _
) As Boolean
```

[C#]

```
public bool ExportMessage(
   string fileName,
   MimeExportOptions options
);
```

Parameters

fileName
A string which specifies the name of the file that will contain the message. If the file does not exist, it will be created. If it does exist, it will be overwritten with the contents of the message.

options
A MimeExportOptions enumeration value which specifies one or more export options.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ExportMessage** method writes the current message to a file. If the file does not exist, it will be created. If it does exist, it will be overwritten with the contents of the message.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.ExportMessage Overload List

InternetMail.ExtractAllFiles Method

Extract all file attachments from the current message.

Overload List

Extract all file attachments from the current message.

public int ExtractAllFiles();

Extract all file attachments from the current message.

public int ExtractAllFiles(string);

See Also

InternetMail Class | SocketTools Namespace

InternetMail.ExtractAllFiles Method ()

Extract all file attachments from the current message.

[Visual Basic]

```
Overloads Public Function ExtractAllFiles() As Integer
```

[C#]

```
public int ExtractAllFiles();
```

Return Value

This method returns an integer value. If the method succeeds, the return value is the number of attachments that were extracted from the message. A return value of zero indicates that the message did not contain any file attachments. If the method faile, the return value is -1. To get extended error information, check the value of the **LastError** property.

Remarks

This method will extract all of the files that are attached to the current message and store them in the current directory on the local system. If a file with the same name as the attachment already exists, it will be overwritten with the contents of the attachment. If the file attachment was encoded using base64 or uuencode, this method will automatically decode the contents of the attachment.

To store a file attachment on the local system using a name that is different than the file name of the attachment, use the **ExtractFile** method.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.ExtractAllFiles Overload List

InternetMail.ExtractAllFiles Method (String)

Extract all file attachments from the current message.

[Visual Basic]

```
Overloads Public Function ExtractAllFiles( _
   ByVal pathName As String _
) As Integer
```

[C#]

```
public int ExtractAllFiles(
   string pathName
);
```

Parameters

pathName

A string that specifies the name of the directory where the file attachments should be stored. If this parameter is omitted or points to an empty string, the attached files will be stored in the current working directory on the local system.

Return Value

This method returns an integer value. If the method succeeds, the return value is the number of attachments that were extracted from the message. A return value of zero indicates that the message did not contain any file attachments. If the method faile, the return value is -1. To get extended error information, check the value of the **LastError** property.

Remarks

This method will extract all of the files that are attached to the current message and store them in the specified directory. The directory must exist and the current user must have the appropriate permissions to create files there. If a file with the same name as the attachment already exists, it will be overwritten with the contents of the attachment. If the file attachment was encoded using base64 or uuencode, this method will automatically decode the contents of the attachment.

To store a file attachment on the local system using a name that is different than the file name of the attachment, use the **ExtractFile** method.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.ExtractAllFiles Overload List

InternetMail.ExtractFile Method

Extract the contents of a file attachment and store it on the local system.

Overload List

Extract the contents of a file attachment and store it on the local system.

public bool ExtractFile(int,string);

Extract the contents of a file attachment and store it on the local system.

public bool ExtractFile(string);

Extract the contents of a file attachment and store it on the local system.

public bool ExtractFile(string,string);

See Also

InternetMail Class | SocketTools Namespace

InternetMail.ExtractFile Method (String)

Extract the contents of a file attachment and store it on the local system.

[Visual Basic]

```
Overloads Public Function ExtractFile( _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool ExtractFile(
   string fileName
);
```

Parameters

fileName

A string which specifies the name of the file that will contain the file attachment. If the file does not exist, it will be created. If it does exist, it will be overwritten with the contents of the attachment.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ExtractFile** method writes the contents of a message part, typically a file attachment, to a file on the local system. This method will automatically decode any binary file attachments.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.ExtractFile Overload List

InternetMail.ExtractFile Method (String, String)

Extract the contents of a file attachment and store it on the local system.

[Visual Basic]

```
Overloads Public Function ExtractFile( _
   ByVal attachName As String, _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool ExtractFile(
   string attachName,
   string fileName
);
```

Parameters

attachName

A string that specifies the name of the file attachment in the current message. This parameter should only specify a base file name; it should not include a file path and cannot be an empty string

fileName

A string which specifies the name of the file on the local system that will contain the file attachment. If the file does not exist, it will be created. If it does exist, it will be overwritten with the contents of the attachment.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This version of the **ExtractFile** method will search the current message for a file attachment that matches the name specified by the ***attachName*** parameter. If an attachment with that name is found, its contents will be stored in the local file specified by the ***fileName*** parameter. If the message does not contain an attachment that matches the name provided, this method will fail.

To search for a file attachment by name, use the **FindAttachment** method.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.ExtractFile Overload List

InternetMail.ExtractFile Method (Int32, String)

Extract the contents of a file attachment and store it on the local system.

[Visual Basic]

```
Overloads Public Function ExtractFile( _
   ByVal messagePart As Integer, _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool ExtractFile(
   int messagePart,
   string fileName
);
```

Parameters

messagePart

An integer which specifies the message part.

fileName

A string which specifies the name of the file that will contain the file attachment. If the file does not exist, it will be created. If it does exist, it will be overwritten with the contents of the attachment.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ExtractFile** method writes the contents of a message part, typically a file attachment, to a file on the local system. This method will automatically decode any binary file attachments.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.ExtractFile Overload List

InternetMail.Finalize Method

Destroys an instance of the class, releasing the resources allocated for the current message.

[Visual Basic]

```
Overrides Protected Sub Finalize()
```

[C#]

```
protected override void Finalize();
```

See Also

InternetMail Class | SocketTools Namespace

InternetMail.FindAttachment Method

Search for a specific file attachment in the current message.

```
[Visual Basic]
Public Function FindAttachment( _
   ByVal fileName As String _
) As Integer
```

```
[C#]
public int FindAttachment(
   string fileName
);
```

Parameters

fileName

A string value that specifies the name of the file attachment to search for. This parameter should only specify a base file name; it should not include a file path and cannot be an empty string.

Return Value

An integer value which specifies the message part number that contains the file attachment with a matching name. If the message does not contain a file attachment with the specified name, this method will return -1.

Remarks

The **FindAttachment** method will search the current message for a attachment that matches the specified file name. The search is not case-sensitive, however it must match the attachment file name completely. This method will not match partial file names or names that include wildcard characters.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.GetFirstHeader Method

Return the first header in the current message part.

[Visual Basic]

```
Public Function GetFirstHeader( _
   ByRef headerName As String, _
   ByRef headerValue As String _
) As Boolean
```

[C#]

```
public bool GetFirstHeader(
   ref string headerName,
   ref string headerValue
);
```

Parameters

headerName

A string passed by reference which will contain the name of the first header field when the method returns.

headerValue

A string passed by reference which will contain the value of the first header field when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFirstHeader** method allows an application to enumerate all of the headers in the current message part. If the current message part does not contain any header fields, this method will return **false**.

The current message part is returned by the **Part** property.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.GetHeader Method

Return the value of a header field in the specified message part.

Overload List

Return the value of a header field in the specified message part.

public bool GetHeader(int,string,ref string);

Return the value of a header field in the current message part.

public bool GetHeader(string,ref string);

See Also

InternetMail Class | SocketTools Namespace

InternetMail.GetHeader Method (Int32, String, String)

Return the value of a header field in the specified message part.

[Visual Basic]

```
Overloads Public Function GetHeader( _
   ByVal messagePart As Integer, _
   ByVal headerName As String, _
   ByRef headerValue As String _
) As Boolean
```

[C#]

```
public bool GetHeader(
   int messagePart,
   string headerName,
   ref string headerValue
);
```

Parameters

messagePart
An integer which specifies the message part.

headerName
A string which specifies the name of the header field.

headerValue
A string passed by reference which will contain the value of the header field when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeader** method is used to retrieve the value for a specific header in the specified message part. If the header field exists, the method will return **true** and the ***headerValue*** argument will contain the header value. If the header does not exist, the method will return **false**.

If there are multiple headers with the same name, the first value will be returned. To enumerate all of the headers in a message, including duplicate header fields, use the **GetFirstHeader** and **GetNextHeader** methods.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.GetHeader Overload List

InternetMail.GetHeader Method (String, String)

Return the value of a header field in the current message part.

[Visual Basic]

```
Overloads Public Function GetHeader( _
   ByVal headerName As String, _
   ByRef headerValue As String _
) As Boolean
```

[C#]

```
public bool GetHeader(
   string headerName,
   ref string headerValue
);
```

Parameters

headerName
A string which specifies the name of the header field.

headerValue
A string passed by reference which will contain the value of the header field when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeader** method is used to retrieve the value for a specific header in the current message part. If the header field exists, the method will return **true** and the ***headerValue*** argument will contain the header value. If the header does not exist, the method will return **false**.

If there are multiple headers with the same name, the first value will be returned. To enumerate all of the headers in a message, including duplicate header fields, use the **GetFirstHeader** and **GetNextHeader** methods.

The current message part is returned by the **Part** property.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.GetHeader Overload List

InternetMail.GetHeaders Method

Retrieves the headers for the current message from the server.

Overload List

Retrieves the headers for the current message from the server.

public bool GetHeaders();

Retrieves the headers for the current message from the server.

public bool GetHeaders(byte[],ref int);

Retrieves the headers for the specified message from the server.

public bool GetHeaders(int);

Retrieves the headers for the specified message from the server.

public bool GetHeaders(int,byte[],ref int);

Retrieves the headers for the specified message from the server.

public bool GetHeaders(int,ref string);

Retrieves the headers for the current message from the server.

public bool GetHeaders(ref string);

See Also

InternetMail Class | SocketTools Namespace

InternetMail.GetHeaders Method ()

Retrieves the headers for the current message from the server.

[Visual Basic]

```
Overloads Public Function GetHeaders() As Boolean
```

[C#]

```
public bool GetHeaders();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeaders** method is used to retrieve the message headers from the server, replacing the contents of the current message with the header values. This method will cause the current thread to block until the article transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

This method will clear the contents of the current message, including the body of the current message text, and replace it with the header values returned by the server.

Note that the header data will be from the first part of the message, not from any additional sections of a multipart message. In other words, the headers such as From, To, Subject and Date will be returned in the buffer.

The current message number is specified by the value of the **Message** property.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.GetHeaders Overload List

InternetMail.GetHeaders Method (Byte[], Int32)

Retrieves the headers for the current message from the server.

[Visual Basic]

```
Overloads Public Function GetHeaders( _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool GetHeaders(
   byte[] buffer,
   ref int length
);
```

Parameters

buffer
A byte array that will contain the message data when the method returns.

length
An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeaders** method is used to retrieve the message headers from the server and copy it into a local buffer. This method will cause the current thread to block until the article transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Note that the header data will be from the first part of the message, not from any additional sections of a multipart message. In other words, the headers such as From, To, Subject and Date will be returned in the buffer.

The current message number is specified by the value of the **Message** property.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.GetHeaders Overload List

InternetMail.GetHeaders Method (Int32)

Retrieves the headers for the specified message from the server.

[Visual Basic]

```
Overloads Public Function GetHeaders( _
   ByVal messageId As Integer _
) As Boolean
```

[C#]

```
public bool GetHeaders(
   int messageId
);
```

Parameters

messageId

Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeaders** method is used to retrieve the message headers from the server, replacing the contents of the current message with the header values. This method will cause the current thread to block until the article transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

This method will clear the contents of the current message, including the body of the current message text, and replace it with the header values returned by the server.

Note that the header data will be from the first part of the message, not from any additional sections of a multipart message. In other words, the headers such as From, To, Subject and Date will be returned in the buffer.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.GetHeaders Overload List

InternetMail.GetHeaders Method (Int32, Byte[], Int32)

Retrieves the headers for the specified message from the server.

[Visual Basic]

```
Overloads Public Function GetHeaders( _
   ByVal messageId As Integer, _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool GetHeaders(
   int messageId,
   byte[] buffer,
   ref int length
);
```

Parameters

messageId
Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

buffer
A byte array that will contain the message data when the method returns.

length
An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeaders** method is used to retrieve the message headers from the server and copy it into a local buffer. This method will cause the current thread to block until the article transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Note that the header data will be from the first part of the message, not from any additional sections of a multipart message. In other words, the headers such as From, To, Subject and Date will be returned in the buffer.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.GetHeaders Overload List

InternetMail.GetHeaders Method (Int32, String)

Retrieves the headers for the specified message from the server.

[Visual Basic]

```
Overloads Public Function GetHeaders( _
   ByVal messageId As Integer, _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool GetHeaders(
   int messageId,
   ref string buffer
);
```

Parameters

messageId

Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

buffer

A string passed by reference that will contain the message data when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeaders** method is used to retrieve the message headers from the server and copy it into a local buffer. This method will cause the current thread to block until the article transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Note that the header data will be from the first part of the message, not from any additional sections of a multipart message. In other words, the headers such as From, To, Subject and Date will be returned in the buffer.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.GetHeaders Overload List

InternetMail.GetHeaders Method (String)

Retrieves the headers for the current message from the server.

[Visual Basic]

```
Overloads Public Function GetHeaders( _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool GetHeaders(
   ref string buffer
);
```

Parameters

buffer
A string passed by reference that will contain the message data when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeaders** method is used to retrieve the message headers from the server and copy it into a local buffer. This method will cause the current thread to block until the article transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Note that the header data will be from the first part of the message, not from any additional sections of a multipart message. In other words, the headers such as From, To, Subject and Date will be returned in the buffer. To retrieve the headers from a specific section of a multipart message, you can use the **GetMessage** method and specify the **ImapSections.sectionHeader** option.

The current message number is specified by the value of the **Message** property.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.GetHeaders Overload List

InternetMail.GetMessage Method

Retrieve a message from the server and replace the current message with its contents.

Overload List

Retrieve a message from the server and replace the current message with its contents.

public bool GetMessage();

Retrieve the current message from the server and return the contents in a byte array.

public bool GetMessage(byte[],ref int);

Retrieve a message from the server and replace the current message with its contents.

public bool GetMessage(int);

Retrieve a message from the server and return the contents in a byte array.

public bool GetMessage(int,byte[],ref int);

Retrieve a message from the server and return the contents in a string.

public bool GetMessage(int,int,ref string);

Retrieve a message from the server and return the contents in a string.

public bool GetMessage(int,int,ref string,ImapSections);

Retrieve a message from the server and return the contents in a string.

public bool GetMessage(int,ref string);

Retrieve a message from the server and return the contents in a string.

public bool GetMessage(int,ref string,ImapSections);

Retrieve the current message from the server and return the contents in a string.

public bool GetMessage(ref string);

See Also

InternetMail Class | SocketTools Namespace

InternetMail.GetMessage Method ()

Retrieve a message from the server and replace the current message with its contents.

[Visual Basic]

```
Overloads Public Function GetMessage() As Boolean
```

[C#]

```
public bool GetMessage();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve a message from the server and replace the current message with its contents. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The current message number is specified by the value of the **Message** property.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.GetMessage Overload List

InternetMail.GetMessage Method (Byte[], Int32)

Retrieve the current message from the server and return the contents in a byte array.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool GetMessage(
   byte[] buffer,
   ref int length
);
```

Parameters

buffer

A byte array that the message data will be stored in.

length

An integer value passed by reference which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller. When the method returns, this parameter will be updated with the actual number of bytes copied into the array.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve the current message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.GetMessage Overload List

InternetMail.GetMessage Method (Int32)

Retrieve a message from the server and replace the current message with its contents.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByVal messageId As Integer _
) As Boolean
```

[C#]

```
public bool GetMessage(
   int messageId
);
```

Parameters

messageId

Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve the specified message from the server and replace the current message with its contents. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The current message number is specified by the value of the **Message** property.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.GetMessage Overload List

InternetMail.GetMessage Method (Int32, Byte[], Int32)

Retrieve a message from the server and return the contents in a byte array.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByVal messageId As Integer, _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool GetMessage(
   int messageId,
   byte[] buffer,
   ref int length
);
```

Parameters

messageId
Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

buffer
A byte array that the message data will be stored in.

length
An integer value passed by reference which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller. When the method returns, this parameter will be updated with the actual number of bytes copied into the array.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve a message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.GetMessage Overload List

InternetMail.GetMessage Method (Int32, Int32, String)

Retrieve a message from the server and return the contents in a string.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByVal messageId As Integer, _
   ByVal messagePart As Integer, _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool GetMessage(
   int messageId,
   int messagePart,
   ref string buffer
);
```

Parameters

messageId
Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

messagePart
An integer value that specifies the message part that should be retrieved. A value of zero specifies that the complete message should be returned. If the message is a multipart MIME message, message parts start with a value of one.

buffer
A string passed by reference that will contain the message data when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve a message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.GetMessage Overload List

InternetMail.GetMessage Method (Int32, Int32, String, ImapSections)

Retrieve a message from the server and return the contents in a string.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByVal messageId As Integer, _
   ByVal messagePart As Integer, _
   ByRef buffer As String, _
   ByVal options As ImapSections _
) As Boolean
```

[C#]

```
public bool GetMessage(
   int messageId,
   int messagePart,
   ref string buffer,
   ImapSections options
);
```

Parameters

messageId
Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

messagePart
An integer value that specifies the message part that should be retrieved. A value of zero specifies that the complete message should be returned. If the message is a multipart MIME message, message parts start with a value of one.

buffer
A string passed by reference that will contain the message data when the method returns.

options
An ImapSections enumeration value which specifies which section of the message to return.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve a message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.GetMessage Overload List

InternetMail.GetMessage Method (Int32, String)

Retrieve a message from the server and return the contents in a string.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByVal messageId As Integer, _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool GetMessage(
   int messageId,
   ref string buffer
);
```

Parameters

messageId

Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

buffer

A string passed by reference that will contain the message data when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve a message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.GetMessage Overload List

InternetMail.GetMessage Method (Int32, String, ImapSections)

Retrieve a message from the server and return the contents in a string.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByVal messageId As Integer, _
   ByRef buffer As String, _
   ByVal options As ImapSections _
) As Boolean
```

[C#]

```
public bool GetMessage(
   int messageId,
   ref string buffer,
   ImapSections options
);
```

Parameters

messageId
Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

buffer
A string passed by reference that will contain the message data when the method returns.

options
An ImapSections enumeration value which specifies which section of the message to return.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve a message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.GetMessage Overload List

InternetMail.GetMessage Method (String)

Retrieve the current message from the server and return the contents in a string.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool GetMessage(
   ref string buffer
);
```

Parameters

buffer

A string passed by reference that will contain the message data when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve the current message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The current message number is specified by the value of the **Message** property.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.GetMessage Overload List

InternetMail.GetNextHeader Method

Return the next header in the current message part.

[Visual Basic]

```
Public Function GetNextHeader( _
   ByRef headerName As String, _
   ByRef headerValue As String _
) As Boolean
```

[C#]

```
public bool GetNextHeader(
   ref string headerName,
   ref string headerValue
);
```

Parameters

headerName

A string passed by reference which will contain the name of the first header field when the method returns.

headerValue

A string passed by reference which will contain the value of the first header field when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetNextHeader** method allows an application to enumerate all of the headers in the current message part. If the current message part does not contain any header fields, this method will return **false**.

The current message part is returned by the **Part** property.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Idle Method

Enables mailbox status monitoring for the client session.

Overload List

Enables mailbox status monitoring for the client session.

public bool Idle();

Enables mailbox status monitoring for the client session.

public bool Idle(IdleOptions,int);

See Also

InternetMail Class | SocketTools Namespace | OnUpdate

InternetMail.Idle Method ()

Enables mailbox status monitoring for the client session.

[Visual Basic]

```
Overloads Public Function Idle() As Boolean
```

[C#]

```
public bool Idle();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

Many IMAP servers support the ability to asynchronously send status updates to the client, rather than have the client periodically poll the server. The client enables this feature by calling the **Idle** method and providing an event handler for the **OnUpdate** event. Typically these events inform the client that a new message has arrived or that a message has been expunged from the mailbox.

The **Idle** method begins monitoring the client session asynchronously and returns control immediately to the caller. If the server sends a update notification to the client, the **OnUpdate** event will fire with information about the status change. Sending an IMAP command to the server will cause the client to stop monitoring the session for status changes. To explicitly stop monitoring the session, use the **Cancel** method.

This method works by sending the IDLE command to the server and starting a worker thread which monitors the connection and looks for untagged responses issued by the server. Events will be generated for EXISTS, EXPUNGE and RECENT messages. Note that some servers may periodically send untagged OK messages to the client, indicating that the connection is still active; these messages are explicitly ignored.

The **OnUpdate** event is invoked within the context of the worker thread that is monitoring the client session. Because of this, applications should not directly update the user interface from within the event handler. For example, if the server sends a notification that a new email message has arrived, the application should not attempt to read the new message and update the user interface directly from within the event handler. Instead, it should create a delegate and use the **Control.Invoke** method to marshal the call to the thread that owns the control's window handle. Failure to do this can cause the application to become unstable. For more information, refer to the **Control.Invoke** method in the .NET Framework documentation.

An application should never make an assumption about how a particular server may send update notifications to the client. Servers can be configured to use different intervals at which notifications are sent. For example, a server may send new message notifications immediately, but may periodically notify the client when a message has been expunged. Alternatively, a server may only send notifications at fixed intervals, in which case the client would not be notified of any new messages until the interval period is reached. It is not possible for a client to know what a particular server's update interval is. Applications that require that degree of control should not use the **Idle** method and should poll the server instead.

This method can only be used when connected to an IMAP server. Attempting to use this method when connected to a POP3 server will cause the method to fail, returning an error indicating that the feature is not supported.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.Idle Overload List | OnUpdate

InternetMail.Idle Method (IdleOptions, Int32)

Enables mailbox status monitoring for the client session.

[Visual Basic]

```
Overloads Public Function Idle( _
   ByVal options As IdleOptions, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Idle(
   IdleOptions options,
   int timeout
);
```

Parameters

options

One or more of the IdleOptions enumeration flags.

timeout

Specifies the timeout period in seconds to wait for a notification from the server. This parameter is only used when the **ImapIdle.idleWait** option has been specified.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

Many IMAP servers support the ability to asynchronously send status updates to the client, rather than have the client periodically poll the server. The client enables this feature by calling the **Idle** method and providing an event handler for the **OnUpdate** event. Typically these events inform the client that a new message has arrived or that a message has been expunged from the mailbox.

The **Idle** method can operate in one of two modes, based on the options specified by the caller. If the option **idleNoWait** is specified, the method begins monitoring the client session asynchronously and returns control immediately to the caller. If the server sends a update notification to the client, the **OnUpdate** event will fire with information about the status change. If the option **idleWait** is specified, the method will block waiting for the server to send a notification message to the client. The method will return when either a message is received or the timeout period is exceeded. Sending an IMAP command to the server will cause the client to stop monitoring the session for status changes. To explicitly stop monitoring the session, use the **Cancel** method.

This method works by sending the IDLE command to the server and starting a worker thread which monitors the connection and looks for untagged responses issued by the server. Events will be generated for EXISTS, EXPUNGE and RECENT messages. Note that some servers may periodically send untagged OK messages to the client, indicating that the connection is still active; these messages are explicitly ignored.

The **OnUpdate** event is invoked within the context of the worker thread that is monitoring the client session. Because of this, applications should not directly update the user interface from within the event handler. For example, if the server sends a notification that a new email message has arrived, the application should not attempt to read the new message and update the user interface directly from within the event handler. Instead, it should create a delegate and use the **Control.Invoke** method to marshal the call to the thread that owns the control's window handle. Failure to do this can cause the application to become unstable. For more information, refer to the **Control.Invoke** method in the .NET Framework documentation.

An application should never make an assumption about how a particular server may send update notifications to the client. Servers can be configured to use different intervals at which notifications are sent. For example, a server may send new message notifications immediately, but may periodically notify the client when a message has been expunged. Alternatively, a server may only send notifications at fixed intervals, in which case the client would not be notified of any new messages until the interval period is reached. It is not possible for a client to know what a particular server's update interval is. Applications that require that degree of control should not use the **Idle** method and should poll the server instead.

This method can only be used when connected to an IMAP server. Attempting to use this method when connected to a POP3 server will cause the method to fail, returning an error indicating that the feature is not supported.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.Idle Overload List | OnUpdate

InternetMail.ImportMessage Method

Replace the current message with the contents of a file.

[Visual Basic]

```
Public Function ImportMessage( _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool ImportMessage(
   string fileName
);
```

Parameters

fileName

A string which specifies the name of the text file to import.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Initialize Method

Initialize an instance of the InternetMail class.

Overload List

Initialize an instance of the InternetMail class.

public bool Initialize();

Initialize an instance of the InternetMail class.

public bool Initialize(string);

See Also

InternetMail Class | SocketTools Namespace | Uninitialize Method

InternetMail.Initialize Method ()

Initialize an instance of the InternetMail class.

[Visual Basic]

```
Overloads Public Function Initialize() As Boolean
```

[C#]

```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the InternetMail class, allocating resources for the current thread. Typically it is not necessary to explicitly call this method because the instance of the class is initialized by the class constructor. However, if the **Uninitialize** method is called, the class must be re-initialized before any other methods are called.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.Initialize Overload List | Uninitialize Method

InternetMail.Initialize Method (String)

Initialize an instance of the InternetMail class.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the InternetMail class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the InternetMail class.

Example

The following example shows how to use the Initialize method to initialize an instance of the class. This example assumes that the license key string has been defined in code.

```
SocketTools.InternetMail mimeClient = new SocketTools.InternetMail();

if (mimeClient.Initialize(strLicenseKey) == false)
{
    MessageBox.Show(mimeClient.LastErrorString, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    return;
}
```

```
Dim mimeClient As New SocketTools.InternetMail

If mimeClient.Initialize(strLicenseKey) = False Then
    MsgBox(mimeClient.LastErrorString, vbIconExclamation)
    Exit Sub
End If
```

See Also

InternetMail Class | SocketTools Namespace | InternetMail.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

InternetMail.ParseAddress Method

Parse an Internet email address.

Overload List

Parse an Internet email address.

public bool ParseAddress(string,string,ref string);

Parse an Internet email address.

public bool ParseAddress(string,ref string);

See Also

InternetMail Class | SocketTools Namespace

InternetMail.ParseAddress Method (String, String, String)

Parse an Internet email address.

[Visual Basic]

```
Overloads Public Function ParseAddress( _
   ByVal mailAddress As String, _
   ByVal mailDomain As String, _
   ByRef parsedAddress As String _
) As Boolean
```

[C#]

```
public bool ParseAddress(
   string mailAddress,
   string mailDomain,
   ref string parsedAddress
);
```

Parameters

mailAddress
A string which specifies the email address to be parsed.

mailDomain
A string which specifies a default domain for the address if no domain name is specified in the *mailAddress* parameter.

parsedAddress
A string passed by reference which will contain the parsed email address.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ParseAddress** method is useful for parsing the email addresses that may be specified in various header fields in the message. In many cases, the addresses have additional comment characters which are not part of the address itself. For example, one common format is "User Name" <user@domain.com>. In this case, the email address is enclosed in angle brackets and the name outside of the brackets is considered to be a comment which is not part of the address itself.

Another common format is user@domain.com (User Name). In this case, there is the address followed by a comment which is enclosed in parenthesis. The **ParseAddress** method recognizes both formats, and when passed either string, would return the address user@domain.com.

If there was no domain specified in the address, that is just a user name was specified, then the value the *mailDomain* parameter is added to the address.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.ParseAddress Overload List

InternetMail.ParseAddress Method (String, String)

Parse an Internet email address.

```
[Visual Basic]
Overloads Public Function ParseAddress( _
   ByVal mailAddress As String, _
   ByRef parsedAddress As String _
) As Boolean
```

```
[C#]
public bool ParseAddress(
   string mailAddress,
   ref string parsedAddress
);
```

Parameters

mailAddress
A string which specifies the email address to be parsed.

parsedAddress
A string passed by reference which will contain the parsed email address.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ParseAddress** method is useful for parsing the email addresses that may be specified in various header fields in the message. In many cases, the addresses have additional comment characters which are not part of the address itself. For example, one common format is "User Name" <user@domain.com>. In this case, the email address is enclosed in angle brackets and the name outside of the brackets is considered to be a comment which is not part of the address itself.

Another common format is user@domain.com (User Name). In this case, there is the address followed by a comment which is enclosed in parenthesis. The **ParseAddress** method recognizes both formats, and when passed either string, would return the address user@domain.com.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.ParseAddress Overload List

InternetMail.ParseMessage Method

Parse the specified string, adding the contents to the current message.

[Visual Basic]

```
Public Function ParseMessage( _
   ByVal messageText As String _
) As Boolean
```

[C#]

```
public bool ParseMessage(
   string messageText
);
```

Parameters

messageText

A string which contains the message text to be parsed.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ParseMessage** method parses a string which contains message data, adding it to the current message. This method is useful when the application needs to parse an arbitrary block of text and add it to the current message. If the string contains header fields, the values will be added to the message header. Once the end of the header block is detected, all subsequent text is added to the body of the message.

Note that unlike the **ImportMessage** method, the **ParseMessage** method does not clear the contents of the current message and may be called multiple times. Use the **ClearMessage** method to clear the current message before calling **ParseMessage** if necessary.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.RenameMailbox Method

Change the name of a mailbox.

[Visual Basic]

```
Public Function RenameMailbox( _
   ByVal oldMailbox As String, _
   ByVal newMailbox As String _
) As Boolean
```

[C#]

```
public bool RenameMailbox(
   string oldMailbox,
   string newMailbox
);
```

Parameters

oldMailbox
A string that specifies the name of the mailbox to be renamed on the server. The mailbox must exist on the server, otherwise an error will be returned.

newMailbox
A string that specifies the new name for the mailbox. An error will be returned if a mailbox with that name already exists.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

If the existing mailbox name contains inferior hierarchical names (mailboxes under the specified mailbox) then those mailboxes will also be renamed. For example, if the mailbox "Mail/Pictures" contains two mailboxes, "Personal" and "Work" and it is renamed to "Mail/Images" then the two mailboxes under it would be automatically renamed to "Mail/Images/Personal" and "Mail/Images/Work".

If the mailbox being renamed is the currently selected mailbox, the current mailbox will be unselected and any messages marked for deletion will be expunged. The new mailbox name will then automatically be re-selected. To prevent deleted messages from being removed from the mailbox prior to being renamed, use the **UnselectMailbox** method to unselect the current mailbox before calling **RenameMailbox**. Note that if the rename operation fails, the client may be left in an unselected state.

It is permitted to rename the special mailbox INBOX. In this case, the messages will be moved from the INBOX mailbox to the new mailbox. If the INBOX mailbox is currently selected, the new mailbox will not automatically be selected. INBOX will remain the selected mailbox.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.ReselectMailbox Method

Reselects the current mailbox.

[Visual Basic]

```
Public Function ReselectMailbox() As Boolean
```

[C#]

```
public bool ReselectMailbox();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ReselectMailbox** method forces the current mailbox to be reselected and updates those properties which return information about the mailbox, such as the **MailboxFlags** property. Deleted messages are not expunged from the mailbox and remain marked for deletion.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. All properties will be reset to their default values.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.SearchMailbox Method

Search the current mailbox for messages that match the specified criteria.

Overload List

Search the current mailbox for messages that match the specified criteria.

public int SearchMailbox(string,int[],int);

Search the current mailbox for messages that match the specified criteria and character set.

public int SearchMailbox(string,string,int[],int);

See Also

InternetMail Class | SocketTools Namespace

InternetMail.SearchMailbox Method (String, Int32[], Int32)

Search the current mailbox for messages that match the specified criteria.

[Visual Basic]

```
Overloads Public Function SearchMailbox( _
   ByVal criteria As String, _
   ByVal messageList As Integer(), _
   ByVal maxMessages As Integer _
) As Integer
```

[C#]

```
public int SearchMailbox(
   string criteria,
   int[] messageList,
   int maxMessages
);
```

Parameters

criteria

A string which consists of one or more keywords which are used to define the search criteria. The following keywords are recognized:

Keyword	Description
ANSWERED	Match those messages which have the **ImapFlags.flagAnswered** flag set.
BCC *address*	Match those messages which contain the specified address in the BCC header field.
BEFORE *date*	Match those messages which were added to the mailbox prior to the specified date.
BODY *string*	Match those messages where the body contains the specified string.
CC *address*	Match those messages which contain the specified address in the CC header field.
DELETED	Match those messages which have the **ImapFlags.flagDeleted** flag set.
DRAFT	Match those messages which have the **ImapFlags.flagDraft** flag set.
FLAGGED	Match those messages which have the **ImapFlags.flagUrgent** flag set.
FROM *address*	Match those messages which contain the specified address in the FROM header field.
HEADER *field string*	Match those messages which contain the string in the specified header field. If no string is specified, then all messages which contain the header will be matched.
LARGER *size*	Match those messages which are larger than the

	specified size in bytes.
NEW	Match those messages which have the **ImapFlags.flagRecent** flag set, but not the **ImapFlags.flagSeen** flag.
OLD	Match those messages which do not have the **ImapFlags.flagRecent** flag set.
ON *date*	Match those messages which were added on the specified date.
RECENT	Match those messages which have the **ImapFlags.flagRecent** flag set.
SEEN	Match those messages which have the **ImapFlags.flagSeen** flag set.
SENTBEFORE *date*	Match those messages whose Date header value is earlier than the specified date.
SENTON *date*	Match those messages whose Date header value is the same as the specified date.
SENTSINCE *date*	Match those messages whose Date header value is later than the specified date.
SINCE *date*	Match those messages added to the mailbox after the specified date.
SMALLER *size*	Match those messages which are smaller than the specified size in bytes.
SUBJECT *string*	Match those messages whose Subject header contains the specified string.
TEXT *string*	Match those messages whose headers or body contains the specified string.
TO *address*	Match those messages which contain the specified address in the TO header field.
UID *sequence*	Match those messages with unique identifiers in the sequence set.
UNANSWERED	Match those messages which do not have the **ImapFlags.flagAnswered** flag set.
UNDELETED	Match those messages which do not have the **ImapFlags.flagDeleted** flag set.
UNDRAFT	Match those messages which do not have the **ImapFlags.flagDraft** flag set.
UNFLAGGED	Match those messages which do not have the **ImapFlags.flagUrgent** flag set.
UNSEEN	Match those messages which do not have the **ImapFlags.flagUnseen** flag set.

messageList

An array of integers which will contain the message numbers of those messages which match the search criteria.

maxMessages

An integer value which specifies the maximum number of message numbers which can be returned in the ***messageList*** array. This value cannot be larger than the size of the array.

Return Value

The number of messages which were found to match the search criteria. If no messages match the criteria, then the return value will be zero. A return value of -1 indicates an error, and the specific error code can be determined by checking the value of the **LastError** property.

Remarks

The **SearchMailbox** method is used to search a mailbox for messages which match a given criteria and return a list of the matching message numbers. The search criteria is composed of one or more search keywords and and optional value to match against. String searches are not case sensitive and partial matches in the message are returned. The message numbers returned by this method are only valid until the mailbox is expunged or another mailbox is selected.

In addition to the listed keywords, the keyword NOT may prefix a keyword to return those messages which do not match the search criteria. For example, "NOT TO user@domain.com" would return those messages which were not addressed to user@domain.com.

If multiple search keywords are specified, the result is the intersection of all those messages which meet the search criteria. For example, a search criteria of "DELETED SINCE 1-Jan-2003" would return all those messages which are marked for deletion and were added to the mailbox after 1 January 2003.

Those search keywords which expect dates must be specified in format ***dd-mmm-yyyy*** where the month is the three letter abbreviation for the month name. Note that the internal date the message was added to the mailbox is not the same as the value of the Date header field in the message.

If the search keyword requires a string value and the string contains one or more spaces, you must enclose the search string in quotes as part of the criteria string. The quotes around the search string prevents the server from interpreting it as a multiple search criteria to be evaluated. If you are using a search string provided by a user, it is recommended that you always enclose it in quotes to prevent any potential ambiguity in the search. Even if the search string does not contain any spaces, it is always safe to enclose it in quotes.

The UID keyword expects a one or more unique message identifiers. These values may provided as comma separated list, or a range delimited by a colon. For example, "UID 23000:24000" would return all those messages who have UIDs ranging from 23000 through to 24000.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.SearchMailbox Overload List

InternetMail.SearchMailbox Method (String, String, Int32[], Int32)

Search the current mailbox for messages that match the specified criteria and character set.

[Visual Basic]

```
Overloads Public Function SearchMailbox( _
   ByVal criteria As String, _
   ByVal charset As String, _
   ByVal messageList As Integer(), _
   ByVal maxMessages As Integer _
) As Integer
```

[C#]

```
public int SearchMailbox(
   string criteria,
   string charset,
   int[] messageList,
   int maxMessages
);
```

Parameters

criteria

A string which consists of one or more keywords which are used to define the search criteria. The following keywords are recognized:

Keyword	Description
ANSWERED	Match those messages which have the **ImapFlags.flagAnswered** flag set.
BCC *address*	Match those messages which contain the specified address in the BCC header field.
BEFORE *date*	Match those messages which were added to the mailbox prior to the specified date.
BODY *string*	Match those messages where the body contains the specified string.
CC *address*	Match those messages which contain the specified address in the CC header field.
DELETED	Match those messages which have the **ImapFlags.flagDeleted** flag set.
DRAFT	Match those messages which have the **ImapFlags.flagDraft** flag set.
FLAGGED	Match those messages which have the **ImapFlags.flagUrgent** flag set.
FROM *address*	Match those messages which contain the specified address in the FROM header field.
HEADER *field string*	Match those messages which contain the string in the specified header field. If no string is specified, then all messages which contain the header will be matched.

LARGER *size*	Match those messages which are larger than the specified size in bytes.
NEW	Match those messages which have the **ImapFlags.flagRecent** flag set, but not the **ImapFlags.flagSeen** flag.
OLD	Match those messages which do not have the **ImapFlags.flagRecent** flag set.
ON *date*	Match those messages which were added on the specified date.
RECENT	Match those messages which have the **ImapFlags.flagRecent** flag set.
SEEN	Match those messages which have the **ImapFlags.flagSeen** flag set.
SENTBEFORE *date*	Match those messages whose Date header value is earlier than the specified date.
SENTON *date*	Match those messages whose Date header value is the same as the specified date.
SENTSINCE *date*	Match those messages whose Date header value is later than the specified date.
SINCE *date*	Match those messages added to the mailbox after the specified date.
SMALLER *size*	Match those messages which are smaller than the specified size in bytes.
SUBJECT *string*	Match those messages whose Subject header contains the specified string.
TEXT *string*	Match those messages whose headers or body contains the specified string.
TO *address*	Match those messages which contain the specified address in the TO header field.
UID *sequence*	Match those messages with unique identifiers in the sequence set.
UNANSWERED	Match those messages which do not have the **ImapFlags.flagAnswered** flag set.
UNDELETED	Match those messages which do not have the **ImapFlags.flagDeleted** flag set.
UNDRAFT	Match those messages which do not have the **ImapFlags.flagDraft** flag set.
UNFLAGGED	Match those messages which do not have the **ImapFlags.flagUrgent** flag set.
UNSEEN	Match those messages which do not have the **ImapFlags.flagUnseen** flag set.

charset

An string which specifies the character set to use when searching the mailbox. If this argument is omitted, the default UTF-8 character set will be used. Note that not all servers support searches using anything but the default character set.

messageList

An array of integers which will contain the message numbers of those messages which match the search criteria.

maxMessages

An integer value which specifies the maximum number of message numbers which can be returned in the ***messageList*** array. This value cannot be larger than the size of the array.

Return Value

The number of messages which were found to match the search criteria. If no messages match the criteria, then the return value will be zero. A return value of -1 indicates an error, and the specific error code can be determined by checking the value of the **LastError** property.

Remarks

The **SearchMailbox** method is used to search a mailbox for messages which match a given criteria and return a list of the matching message numbers. The search criteria is composed of one or more search keywords and and optional value to match against. String searches are not case sensitive and partial matches in the message are returned. The message numbers returned by this method are only valid until the mailbox is expunged or another mailbox is selected.

In addition to the listed keywords, the keyword NOT may prefix a keyword to return those messages which do not match the search criteria. For example, "NOT TO user@domain.com" would return those messages which were not addressed to user@domain.com.

If multiple search keywords are specified, the result is the intersection of all those messages which meet the search criteria. For example, a search criteria of "DELETED SINCE 1-Jan-2003" would return all those messages which are marked for deletion and were added to the mailbox after 1 January 2003.

Those search keywords which expect dates must be specified in format ***dd-mmm-yyyy*** where the month is the three letter abbreviation for the month name. Note that the internal date the message was added to the mailbox is not the same as the value of the Date header field in the message.

If the search keyword requires a string value and the string contains one or more spaces, you must enclose the search string in quotes as part of the criteria string. The quotes around the search string prevents the server from interpreting it as a multiple search criteria to be evaluated. If you are using a search string provided by a user, it is recommended that you always enclose it in quotes to prevent any potential ambiguity in the search. Even if the search string does not contain any spaces, it is always safe to enclose it in quotes.

The UID keyword expects a one or more unique message identifiers. These values may provided as comma separated list, or a range delimited by a colon. For example, "UID 23000:24000" would return all those messages who have UIDs ranging from 23000 through to 24000.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.SearchMailbox Overload List

InternetMail.SelectMailbox Method

Selects the specified mailbox for read-write access.

[Visual Basic]

```
Public Function SelectMailbox( _
   ByVal mailboxName As String _
) As Boolean
```

[C#]

```
public bool SelectMailbox(
   string mailboxName
);
```

Parameters

mailboxName

A string argument which specifies the name of the mailbox to be selected.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SelectMailbox** method is used to select a mailbox in read-write mode. If the client has a different mailbox currently selected, that mailbox will be closed and any messages marked for deletion will be expunged. To prevent deleted messages from being removed from the previous mailbox, use the **UnselectMailbox** method prior to selecting the new mailbox.

The special case-insensitive mailbox name INBOX is used for new messages. Other mailbox names may or may not be case-sensitive depending on the IMAP server's operating system and implementation.

To access a mailbox in read-only mode, use the **ExamineMailbox** method.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.SendMessage Method

Submit the current message to a mail server for delivery.

Overload List

Submit the current message to a mail server for delivery.

public bool SendMessage();

Submit the current message to a mail server for delivery.

public bool SendMessage(string,int);

Submit the specified message to a mail server for delivery.

public bool SendMessage(string,int,int);

Submit the current message to a mail server for delivery.

public bool SendMessage(string,int,int,InternetMailOptions);

Submit the current message to a mail server for delivery.

public bool SendMessage(string,int,string,string);

Submit the current message to a mail server for delivery.

public bool SendMessage(string,int,string,string,int);

Submit the current message to a mail server for delivery.

public bool SendMessage(string,int,string,string,int,InternetMailOptions);

Submit the specified message to a mail server for delivery.

public bool SendMessage(string,int,string,string,string,string,string,int,InternetMailOptions);

Submit the current message to a mail server for delivery.

public bool SendMessage(string,string);

Submit the current message to a mail server for delivery.

public bool SendMessage(string,string,string);

See Also

InternetMail Class | SocketTools Namespace

InternetMail.SendMessage Method ()

Submit the current message to a mail server for delivery.

[Visual Basic]

```
Overloads Public Function SendMessage() As Boolean
```

[C#]

```
public bool SendMessage();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SendMessage** method enables an application to send the current message directly to the recipient's mail server or through a relay server. The sender's return address is automatically determined by the value of the **From** property. The recipient addresses are automatically determined by the value of the **To**, **Cc** and **Bcc** properties.

This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

For each recipient specified in the message, the **SendMessage** method will determine the appropriate mail exchange server and deliver the message to that user. If the **RelayServer** and **RelayPort** properties are defined, then all messages will be relayed through that specific server, regardless of the recipient address. Note that the **Secure** property and related options only affects connections to relay mail servers. See the **RelayServer** and **RelayPort** properties for additional information.

If a relay server is being used, it may require authentication before accepting any messages for delivery. To enable authentication, specify the **optionAuthLogin** option by setting the **Options** property. Prior to calling the **SendMessage** method, the **UserName** and **Password** properties should be set to the values that will be used to authenticate the session. If the server does not support authentication, or the user name or password is invalid, an error will be returned. Note that authentication is only performed if a relay server is used, otherwise the option is ignored.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.SendMessage Overload List

InternetMail.SendMessage Method (String, Int32)

Submit the current message to a mail server for delivery.

[Visual Basic]

```
Overloads Public Function SendMessage( _
   ByVal hostName As String, _
   ByVal hostPort As Integer _
) As Boolean
```

[C#]

```
public bool SendMessage(
   string hostName,
   int hostPort
);
```

Parameters

hostName

A string value which specifies the host name or IP address of the mail server that the message will be submitted to for delivery.

hostPort

An integer value which specifies the port number which should be used to establish a connection with the mail server. The default port number is 25 for the Simple Mail Transfer Protocol.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SendMessage** method enables an application to send the current email message using the current mail server. The sender's return address will automatically be determined by the value of the **From** property. The recipients for the message will be automatically determined by the value of the **To**, **Cc** and **Bcc** properties.

This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The mail server that is specified must be configured to permit relaying messages, or the submission may fail. In most cases, the server will require that the client authenticate the session with a username and password. Alternatively, some mail servers require that you connect and authenticate with their POP3 service before the SMTP service will accept a message. Consult the documentation for your mail service provider for more information on the requirements for submitting messages for delivery.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.SendMessage Overload List

InternetMail.SendMessage Method (String, Int32, Int32)

Submit the specified message to a mail server for delivery.

[Visual Basic]

```
Overloads Public Function SendMessage( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool SendMessage(
   string hostName,
   int hostPort,
   int timeout
);
```

Parameters

hostName

A string value which specifies the host name or IP address of the mail server that the message will be submitted to for delivery.

hostPort

An integer value which specifies the port number which should be used to establish a connection with the mail server. The default port number is 25 for the Simple Mail Transfer Protocol.

timeout

An integer value which specifies a timeout period in seconds. If the message cannot be submitted within the specified time period, the method will fail. The default timeout value for connections is 20 seconds.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SendMessage** method enables an application to send a formatted email message using the current mail server. This provides a convenient one-step method of addressing and sending a message.

This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The mail server that is specified must be configured to permit relaying messages, or the submission may fail. In most cases, the server will require that the client authenticate the session with a username and password. Alternatively, some mail servers require that you connect and authenticate with their POP3 service before the SMTP service will accept a message. Consult the documentation for your mail service provider for more information on the requirements for submitting messages for delivery.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.SendMessage Overload List

InternetMail.SendMessage Method (String, Int32, Int32, InternetMailOptions)

Submit the current message to a mail server for delivery.

[Visual Basic]

```
Overloads Public Function SendMessage( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer, _
   ByVal options As InternetMailOptions _
) As Boolean
```

[C#]

```
public bool SendMessage(
   string hostName,
   int hostPort,
   int timeout,
   InternetMailOptions options
);
```

Parameters

hostName
A string value which specifies the host name or IP address of the mail server that the message will be submitted to for delivery.

hostPort
An integer value which specifies the port number which should be used to establish a connection with the mail server. The default port number is 25 for the Simple Mail Transfer Protocol.

timeout
An integer value which specifies a timeout period in seconds. If the message cannot be submitted within the specified time period, the method will fail. The default timeout value for connections is 20 seconds.

options
An InternetMailOptions enumeration which specifies the options that can be used when delivering the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SendMessage** method enables an application to send the current email message using the current mail server. The sender's return address will automatically be determined by the value of the **From** property. The recipients for the message will be automatically determined by the value of the **To**, **Cc** and **Bcc** properties.

This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The mail server that is specified must be configured to permit relaying messages, or the submission may fail. In most cases, the server will require that the client authenticate the session with a username and password. Alternatively, some mail servers require that you connect and authenticate with their POP3 service before the SMTP service will accept a message. Consult the documentation for your mail service provider for more information on the requirements for submitting messages for delivery.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.SendMessage Overload List

InternetMail.SendMessage Method (String, Int32, String, String)

Submit the current message to a mail server for delivery.

[Visual Basic]

```
Overloads Public Function SendMessage( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]

```
public bool SendMessage(
   string hostName,
   int hostPort,
   string userName,
   string userPassword
);
```

Parameters

hostName
A string value which specifies the host name or IP address of the mail server that the message will be submitted to for delivery.

hostPort
An integer value which specifies the port number which should be used to establish a connection with the mail server. The default port number is 25 for the Simple Mail Transfer Protocol.

userName
A string value which specifies the user name that will be used to authenticate the client session with the mail server.

userPassword
A string value which specifies the password that will be used to authenticate the client session with the mail server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SendMessage** method enables an application to send the current email message using the current mail server. The sender's return address will automatically be determined by the value of the **From** property. The recipients for the message will be automatically determined by the value of the **To**, **Cc** and **Bcc** properties.

This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The mail server that is specified must be configured to permit relaying messages, or the submission may fail. In most cases, the server will require that the client authenticate the session with a username and password. Alternatively, some mail servers require that you connect and authenticate with their POP3 service before the SMTP service will accept a message. Consult the documentation for your mail service provider for more information on the requirements for submitting messages for delivery.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.SendMessage Overload List

InternetMail.SendMessage Method (String, Int32, String, String, Int32)

Submit the current message to a mail server for delivery.

[Visual Basic]

```
Overloads Public Function SendMessage( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool SendMessage(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   int timeout
);
```

Parameters

hostName

A string value which specifies the host name or IP address of the mail server that the message will be submitted to for delivery.

hostPort

An integer value which specifies the port number which should be used to establish a connection with the mail server. The default port number is 25 for the Simple Mail Transfer Protocol.

userName

A string value which specifies the user name that will be used to authenticate the client session with the mail server.

userPassword

A string value which specifies the password that will be used to authenticate the client session with the mail server.

timeout

An integer value which specifies a timeout period in seconds. If the message cannot be submitted within the specified time period, the method will fail. The default timeout value for connections is 20 seconds.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SendMessage** method enables an application to send the current email message using the current mail server. The sender's return address will automatically be determined by the value of the **From** property. The recipients for the message will be automatically determined by the value of the **To**, **Cc** and **Bcc** properties.

This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The mail server that is specified must be configured to permit relaying messages, or the submission may fail. In most cases, the server will require that the client authenticate the session with a username and password. Alternatively, some mail servers require that you connect and authenticate with their POP3 service before the SMTP service will accept a message. Consult the documentation for your mail service provider for more information on the requirements for submitting messages for delivery.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.SendMessage Overload List

InternetMail.SendMessage Method (String, Int32, String, String, Int32, InternetMailOptions)

Submit the current message to a mail server for delivery.

[Visual Basic]

```
Overloads Public Function SendMessage( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal timeout As Integer, _
   ByVal options As InternetMailOptions _
) As Boolean
```

[C#]

```
public bool SendMessage(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   int timeout,
   InternetMailOptions options
);
```

Parameters

hostName
A string value which specifies the host name or IP address of the mail server that the message will be submitted to for delivery.

hostPort
An integer value which specifies the port number which should be used to establish a connection with the mail server. The default port number is 25 for the Simple Mail Transfer Protocol.

userName
A string value which specifies the user name that will be used to authenticate the client session with the mail server.

userPassword
A string value which specifies the password that will be used to authenticate the client session with the mail server.

timeout
An integer value which specifies a timeout period in seconds. If the message cannot be submitted within the specified time period, the method will fail. The default timeout value for connections is 20 seconds.

options
An InternetMailOptions enumeration which specifies the options that can be used when delivering the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SendMessage** method enables an application to send the current email message using the current mail server. The sender's return address will automatically be determined by the value of the **From** property. The recipients for the message will be automatically determined by the value of the **To**, **Cc** and **Bcc** properties.

This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The mail server that is specified must be configured to permit relaying messages, or the submission may fail. In most cases, the server will require that the client authenticate the session with a username and password. Alternatively, some mail servers require that you connect and authenticate with their POP3 service before the SMTP service will accept a message. Consult the documentation for your mail service provider for more information on the requirements for submitting messages for delivery.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.SendMessage Overload List

InternetMail.SendMessage Method (String, Int32, String, String, String, String, String, Int32, InternetMailOptions)

Submit the specified message to a mail server for delivery.

[Visual Basic]

```
Overloads Public Function SendMessage( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal senderAddress As String, _
   ByVal recipientAddress As String, _
   ByVal messageText As String, _
   ByVal timeout As Integer, _
   ByVal options As InternetMailOptions _
) As Boolean
```

[C#]

```
public bool SendMessage(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   string senderAddress,
   string recipientAddress,
   string messageText,
   int timeout,
   InternetMailOptions options
);
```

Parameters

hostName
A string value which specifies the host name or IP address of the mail server that the message will be submitted to for delivery.

hostPort
An integer value which specifies the port number which should be used to establish a connection with the mail server. The default port number is 25 for the Simple Mail Transfer Protocol.

userName
A string value which specifies the user name that will be used to authenticate the client session with the mail server.

userPassword
A string value which specifies the password that will be used to authenticate the client session with the mail server.

senderAddress
A string argument which specifies the email address of the person sending the message. This typically corresponds to the address in the From header of the message, but it is not required that they be the same.

recipientAddress
A string argument which specifies the email address of the person or persons to receive the message. Multiple addresses may be specified by separating each address with a comma. It should be noted that this protocol is only concerned with the delivery of a message and not its contents. Header fields

in the message are not parsed to automatically determine the recipients. This argument should be a concatenation of all recipients, including carbon copies and blind carbon copies, with each address separated with a comma.

messageText

A string that contains the message to be delivered to the specified recipients. The message must be text and conform to the basic structure defined in RFC 822. There must be one or more headers separated by a blank line, followed by the body of the message. Each line of text must be terminated by a carriage return and linefeed character sequence.

timeout

An integer value which specifies a timeout period in seconds. If the message cannot be submitted within the specified time period, the method will fail. The default timeout value for connections is 20 seconds.

options

An InternetMailOptions enumeration which specifies the options that can be used when delivering the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SendMessage** method enables an application to send a formatted email message using the current mail server. This provides a convenient one-step method of addressing and sending a message.

This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The mail server that is specified must be configured to permit relaying messages, or the submission may fail. In most cases, the server will require that the client authenticate the session with a username and password. Alternatively, some mail servers require that you connect and authenticate with their POP3 service before the SMTP service will accept a message. Consult the documentation for your mail service provider for more information on the requirements for submitting messages for delivery.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.SendMessage Overload List

InternetMail.SendMessage Method (String, String)

Submit the current message to a mail server for delivery.

```
[Visual Basic]
Overloads Public Function SendMessage( _
   ByVal senderAddress As String, _
   ByVal recipientAddress As String _
) As Boolean
```

```
[C#]
public bool SendMessage(
   string senderAddress,
   string recipientAddress
);
```

Parameters

senderAddress

A string argument which specifies the email address of the person sending the message. This typically corresponds to the address in the From header of the message, but it is not required that they be the same.

recipientAddress

A string argument which specifies the email address of the person or persons to receive the message. Multiple addresses may be specified by separating each address with a comma. It should be noted that this protocol is only concerned with the delivery of a message and not its contents. Header fields in the message are not parsed to automatically determine the recipients. This argument should be a concatenation of all recipients, including carbon copies and blind carbon copies, with each address separated with a comma.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SendMessage** method enables an application to send the current message directly to the recipient's mail server or through a relay server.

This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

For each recipient specified, the **SendMessage** method will determine the appropriate mail exchange server and deliver the message to that user. If the **RelayServer** and **RelayPort** properties are defined, then all messages will be relayed through that specific server, regardless of the recipient address. Note that the **Secure** property and related options only affects connections to relay mail servers. See the **RelayServer** and **RelayPort** properties for additional information.

If a relay server is being used, it may require authentication before accepting any messages for delivery. To enable authentication, specify the **optionAuthLogin** option by setting the **Options** property. Prior to calling the **SendMessage** method, the **UserName** and **Password** properties should be set to the values that will be used to authenticate the session. If the server does not support authentication, or the user name or password is invalid, an error will be returned. Note that authentication is only performed if a relay server is used, otherwise the option is ignored.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.SendMessage Overload List

InternetMail.SendMessage Method (String, String, String)

Submit the current message to a mail server for delivery.

```
[Visual Basic]
Overloads Public Function SendMessage( _
   ByVal senderAddress As String, _
   ByVal recipientAddress As String, _
   ByVal messageText As String _
) As Boolean
```

```
[C#]
public bool SendMessage(
   string senderAddress,
   string recipientAddress,
   string messageText
);
```

Parameters

senderAddress
A string argument which specifies the email address of the person sending the message. This typically corresponds to the address in the From header of the message, but it is not required that they be the same.

recipientAddress
A string argument which specifies the email address of the person or persons to receive the message. Multiple addresses may be specified by separating each address with a comma. It should be noted that this protocol is only concerned with the delivery of a message and not its contents. Header fields in the message are not parsed to automatically determine the recipients. This argument should be a concatenation of all recipients, including carbon copies and blind carbon copies, with each address separated with a comma.

messageText
A string that contains the message to be delivered to the specified recipients. The message must be text and conform to the basic structure defined in RFC 822. There must be one or more headers separated by a blank line, followed by the body of the message. Each line of text must be terminated by a carriage return and linefeed character sequence.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SendMessage** method enables an application to send a formatted email message using the current mail server. This provides a convenient one-step method of addressing and sending a message.

This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

For each recipient specified, the **SendMessage** method will determine the appropriate mail exchange server and deliver the message to that user. If the **RelayServer** and **RelayPort** properties are defined, then all messages will be relayed through that specific server, regardless of the recipient address. Note that the **Secure** property and related options only affects connections to relay mail servers. See the

RelayServer and **RelayPort** properties for additional information.

If a relay server is being used, it may require authentication before accepting any messages for delivery. To enable authentication, specify the **optionAuthLogin** option by setting the **Options** property. Prior to calling the **SendMessage** method, the **UserName** and **Password** properties should be set to the values that will be used to authenticate the session. If the server does not support authentication, or the user name or password is invalid, an error will be returned. Note that authentication is only performed if a relay server is used, otherwise the option is ignored.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.SendMessage Overload List

InternetMail.SetHeader Method

Set the value for a header in the specified message part.

Overload List

Set the value for a header in the specified message part.

public bool SetHeader(int,string,string);

Set the value for a header in the current message part.

public bool SetHeader(string,string);

See Also

InternetMail Class | SocketTools Namespace

InternetMail.SetHeader Method (Int32, String, String)

Set the value for a header in the specified message part.

[Visual Basic]

```
Overloads Public Function SetHeader( _
   ByVal messagePart As Integer, _
   ByVal headerName As String, _
   ByVal headerValue As String _
) As Boolean
```

[C#]

```
public bool SetHeader(
   int messagePart,
   string headerName,
   string headerValue
);
```

Parameters

messagePart

An integer which specifies the message part.

headerName

A string which specifies the name of the header field.

headerValue

A string which specifies the value of the header field. If an empty string is specified, the header field is removed from the header block in the specified message part.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.SetHeader Overload List

InternetMail.SetHeader Method (String, String)

Set the value for a header in the current message part.

[Visual Basic]

```
Overloads Public Function SetHeader( _
   ByVal headerName As String, _
   ByVal headerValue As String _
) As Boolean
```

[C#]

```
public bool SetHeader(
   string headerName,
   string headerValue
);
```

Parameters

headerName

A string which specifies the name of the header field.

headerValue

A string which specifies the value of the header field. If an empty string is specified, the header field is removed from the header block in the current message part.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The current message part is returned by the **Part** property.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.SetHeader Overload List

InternetMail.StoreMessage Method

Retrieve a message from the current mailbox and store it in a file on the local system.

Overload List

Retrieve a message from the current mailbox and store it in a file on the local system.

public bool StoreMessage(int,string);

Retrieve the current message and store it in a file on the local system.

public bool StoreMessage(string);

See Also

InternetMail Class | SocketTools Namespace

InternetMail.StoreMessage Method (Int32, String)

Retrieve a message from the current mailbox and store it in a file on the local system.

[Visual Basic]

```
Overloads Public Function StoreMessage( _
   ByVal messageId As Integer, _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool StoreMessage(
   int messageId,
   string fileName
);
```

Parameters

messageId

Number of message to retrieve. This value must be greater than zero. The first message in the mailbox is message number one.

fileName

A string which specifies the file that the message will be stored in. If the file does not exist, it will be created. If the file does exist, it will be overwritten with the contents of the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **StoreMessage** method retrieves a message from the server and stores it in a file on the local system. The contents of the message is stored as a text file, using the specified file name. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.StoreMessage Overload List

InternetMail.StoreMessage Method (String)

Retrieve the current message and store it in a file on the local system.

[Visual Basic]

```
Overloads Public Function StoreMessage( _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool StoreMessage(
   string fileName
);
```

Parameters

fileName

A string which specifies the file that the message will be stored in. If the file does not exist, it will be created. If the file does exist, it will be overwritten with the contents of the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **StoreMessage** method retrieves the current message from the server and stores it in a file on the local system. The contents of the message is stored as a text file, using the specified file name. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The current message number is specified by the value of the **Message** property.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.StoreMessage Overload List

InternetMail.SubscribeMailbox Method

Subscribes the user to the current mailbox.

Overload List

Subscribes the user to the current mailbox.

public bool SubscribeMailbox();

Subscribes the user to the specified mailbox.

public bool SubscribeMailbox(string);

See Also

InternetMail Class | SocketTools Namespace

InternetMail.SubscribeMailbox Method ()

Subscribes the user to the current mailbox.

[Visual Basic]

```
Overloads Public Function SubscribeMailbox() As Boolean
```

[C#]

```
public bool SubscribeMailbox();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SubscribeMailbox** method adds the current mailbox to the current user's list of active or subscribed mailboxes. The user will remain subscribed to the mailbox across multiple sessions, until the **UnsubscribeMailbox** method is called to remove the mailbox from the subscription list.

Note that if a user subscribes to a mailbox and that mailbox is later renamed or deleted, the mailbox will not be automatically removed from the user's subscription list. To determine if the current mailbox is in the user's subscription list, check the **Subscribed** property.

The current mailbox is specified by the value of the **MailboxName** property.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.SubscribeMailbox Overload List

InternetMail.SubscribeMailbox Method (String)

Subscribes the user to the specified mailbox.

[Visual Basic]

```
Overloads Public Function SubscribeMailbox( _
   ByVal mailboxName As String _
) As Boolean
```

[C#]

```
public bool SubscribeMailbox(
   string mailboxName
);
```

Parameters

mailboxName

A string which specifies the name of the mailbox to subscribe to.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SubscribeMailbox** method adds the specified mailbox to the current user's list of active or subscribed mailboxes. The user will remain subscribed to the mailbox across multiple sessions, until the **UnsubscribeMailbox** method is called to remove the mailbox from the subscription list.

Note that if a user subscribes to a mailbox and that mailbox is later renamed or deleted, the mailbox will not be automatically removed from the user's subscription list. To determine if the current mailbox is in the user's subscription list, check the **Subscribed** property.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.SubscribeMailbox Overload List

InternetMail.UndeleteMessage Method

Removes the deletion flag for the specified message.

[Visual Basic]

```
Public Function UndeleteMessage( _
   ByVal messageId As Integer _
) As Boolean
```

[C#]

```
public bool UndeleteMessage(
   int messageId
);
```

Parameters

messageId

Number of message to undelete from the server. This value must be greater than zero. The first message in the mailbox is message number one.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **UndeleteMessage** method removes the deletion flag for the specified message in the current mailbox. To determine if a message has been marked for deletion, set the **Message** property to the message number and then check the value of the **MessageFlags** property to determine if the **imapFlagDeleted** bit flag has been set.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.Uninitialize Method

Uninitialize the class library and release any resources allocated for the current thread.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method releases resources allocated for the current process. After this method has been called, no further operations may be performed until the class instance has been re-initialized.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the class is no longer needed.

See Also

InternetMail Class | SocketTools Namespace | Initialize Method

InternetMail.UnselectMailbox Method

Unselects the current mailbox and expunges deleted messages.

Overload List

Unselects the current mailbox and expunges deleted messages.

public bool UnselectMailbox();

Unselects the current mailbox.

public bool UnselectMailbox(bool);

See Also

InternetMail Class | SocketTools Namespace

InternetMail.UnselectMailbox Method ()

Unselects the current mailbox and expunges deleted messages.

[Visual Basic]

```
Overloads Public Function UnselectMailbox() As Boolean
```

[C#]

```
public bool UnselectMailbox();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **UnselectMailbox** method unselects the current mailbox. Once the mailbox has been unselected, no messages in that mailbox can be accessed, and any messages which have been flagged for deletion are removed.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.UnselectMailbox Overload List

InternetMail.UnselectMailbox Method (Boolean)

Unselects the current mailbox.

[Visual Basic]

```
Overloads Public Function UnselectMailbox( _
   ByVal expunge As Boolean _
) As Boolean
```

[C#]

```
public bool UnselectMailbox(
   bool expunge
);
```

Parameters

expunge

An boolean value which determines if deleted messages will be expunged from the mailbox. A value of **true** specifies that messages that have been marked for deletion will be removed from the mailbox. A value of **false** specifies that no messages will be removed from the mailbox.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **UnselectMailbox** method unselects the current mailbox. Once the mailbox has been unselected, no messages in that mailbox can be accessed.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.UnselectMailbox Overload List

InternetMail.UnsubscribeMailbox Method

Unsubscribes the user from the current mailbox.

Overload List

Unsubscribes the user from the current mailbox.

public bool UnsubscribeMailbox();

Unsubscribes the user from the specified mailbox.

public bool UnsubscribeMailbox(string);

See Also

InternetMail Class | SocketTools Namespace

InternetMail.UnsubscribeMailbox Method ()

Unsubscribes the user from the current mailbox.

[Visual Basic]

```
Overloads Public Function UnsubscribeMailbox() As Boolean
```

[C#]

```
public bool UnsubscribeMailbox();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **UnsubscribeMailbox** method removes the current mailbox from the current user's list of active or subscribed mailboxes. To determine if the current mailbox is in the user's subscription list, check the **Subscribed** property.

The current mailbox is specified by the value of the **MailboxName** property.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.UnsubscribeMailbox Overload List

InternetMail.UnsubscribeMailbox Method (String)

Unsubscribes the user from the specified mailbox.

[Visual Basic]

```
Overloads Public Function UnsubscribeMailbox( _
   ByVal mailboxName As String _
) As Boolean
```

[C#]

```
public bool UnsubscribeMailbox(
   string mailboxName
);
```

Parameters

mailboxName

A string which specifies the name of the mailbox to unsubscribe from.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **UnsubscribeMailbox** method removes the specified mailbox from the current user's list of active or subscribed mailboxes. To determine if the current mailbox is in the user's subscription list, check the **Subscribed** property.

See Also

InternetMail Class | SocketTools Namespace | InternetMail.UnsubscribeMailbox Overload List

InternetMail Events

The events of the **InternetMail** class are listed below. For a complete list of **InternetMail** class members, see the InternetMail Members topic.

Public Instance Events

⚡ OnCancel	Occurs when a blocking client operation is canceled.
⚡ OnDelivered	Occurs when a message has been submitted for delivery.
⚡ OnError	Occurs when an client operation fails.
⚡ OnProgress	Occurs as a data stream is being read or written to the client.
⚡ OnRecipient	Occurs when a message is about to be submitted for delivery.
⚡ OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
⚡ OnUpdate	Occurs when the server sends a mailbox update notification to the client.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.OnCancel Event

Occurs when a blocking client operation is canceled.

[Visual Basic]

```
Public Event OnCancel As EventHandler
```

[C#]

```
public event EventHandler OnCancel;
```

Remarks

The **OnCancel** event is generated when a blocking network operation, such as sending or receiving a message, is canceled with the **Cancel** method. To assist in determining which operation was canceled, check the value of the **Status** property.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.OnDelivered Event

Occurs when a message has been submitted for delivery.

[Visual Basic]
```
Public Event OnDelivered As OnDeliveredEventHandler
```

[C#]
```
public event OnDeliveredEventHandler OnDelivered;
```

Event Data

The event handler receives an argument of type InternetMail.DeliveredEventArgs containing data related to this event. The following **InternetMail.DeliveredEventArgs** properties provide information specific to this event.

Property	Description
Address	Gets the email address of the message recipient.
MessageSize	Gets a value which specifies the total number of bytes in the message.

Remarks

The **OnDelivered** event is generated when a message has been successfully submitted to the mail server for delivery. If multiple recipients have been specified for the message, this event will fire for each recipient, enabling the application to update its user interface as the message is delivered.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.DeliveredEventArgs Class

Provides data for the OnDelivered event.

For a list of all members of this type, see InternetMail.DeliveredEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.InternetMail.DeliveredEventArgs

[Visual Basic]

```
Public Class InternetMail.DeliveredEventArgs
    Inherits EventArgs
```

[C#]

```
public class InternetMail.DeliveredEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

DeliveredEventArgs specifies the address of the message recipient and the size of the message that was delivered in bytes.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

InternetMail.DeliveredEventArgs Members | SocketTools Namespace

InternetMail.DeliveredEventArgs Members

InternetMail.DeliveredEventArgs overview

Public Instance Constructors

InternetMail.DeliveredEventArgs Constructor	Initializes a new instance of the InternetMail.DeliveredEventArgs class.

Public Instance Properties

Address	Gets the email address of the message recipient.
MessageSize	Gets a value which specifies the total number of bytes in the message.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetMail.DeliveredEventArgs Class | SocketTools Namespace

InternetMail.DeliveredEventArgs Constructor

Initializes a new instance of the InternetMail.DeliveredEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public InternetMail.DeliveredEventArgs();
```

See Also

InternetMail.DeliveredEventArgs Class | SocketTools Namespace

InternetMail.DeliveredEventArgs Properties

The properties of the **InternetMail.DeliveredEventArgs** class are listed below. For a complete list of **InternetMail.DeliveredEventArgs** class members, see the InternetMail.DeliveredEventArgs Members topic.

Public Instance Properties

Address	Gets the email address of the message recipient.
MessageSize	Gets a value which specifies the total number of bytes in the message.

See Also

InternetMail.DeliveredEventArgs Class | SocketTools Namespace

InternetMail.DeliveredEventArgs.Address Property

Gets the email address of the message recipient.

[Visual Basic]

```
Public ReadOnly Property Address As String
```

[C#]

```
public string Address {get;}
```

Property Value

A string value which specifies the recipient of the email message.

Remarks

The **Address** property returns the email address of the recipient for the message that was submitted to the mail server for delivery.

See Also

InternetMail.DeliveredEventArgs Class | SocketTools Namespace | MessageSize Property

InternetMail.DeliveredEventArgs.MessageSize Property

Gets a value which specifies the total number of bytes in the message.

[Visual Basic]

```
Public ReadOnly Property MessageSize As Integer
```

[C#]

```
public int MessageSize {get;}
```

Property Value

An integer value which specifies the number of bytes of data.

Remarks

The **MessageSize** property specifies the size of the message submitted for delivery to the mail server.

See Also

InternetMail.DeliveredEventArgs Class | SocketTools Namespace | Address Property

InternetMail.OnError Event

Occurs when an client operation fails.

[Visual Basic]
```
Public Event OnError As OnErrorEventHandler
```

[C#]
```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type InternetMail.ErrorEventArgs containing data related to this event. The following **InternetMail.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a client operation fails.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see InternetMail.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.InternetMail.ErrorEventArgs

[Visual Basic]

```
Public Class InternetMail.ErrorEventArgs
    Inherits EventArgs
```

[C#]

```
public class InternetMail.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

InternetMail.ErrorEventArgs Members | SocketTools Namespace

InternetMail.ErrorEventArgs Members

InternetMail.ErrorEventArgs overview

Public Instance Constructors

InternetMail.ErrorEventArgs Constructor	Initializes a new instance of the InternetMail.ErrorEventArgs class.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetMail.ErrorEventArgs Class | SocketTools Namespace

InternetMail.ErrorEventArgs Constructor

Initializes a new instance of the InternetMail.ErrorEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public InternetMail.ErrorEventArgs();
```

See Also

InternetMail.ErrorEventArgs Class | SocketTools Namespace

InternetMail.ErrorEventArgs Properties

The properties of the **InternetMail.ErrorEventArgs** class are listed below. For a complete list of **InternetMail.ErrorEventArgs** class members, see the InternetMail.ErrorEventArgs Members topic.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

InternetMail.ErrorEventArgs Class | SocketTools Namespace

InternetMail.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

InternetMail.ErrorEventArgs Class | SocketTools Namespace | Error Property

InternetMail.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public InternetMail.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

InternetMail.ErrorEventArgs Class | SocketTools Namespace | Description Property

InternetMail.OnProgress Event

Occurs as a data stream is being read or written to the client.

[Visual Basic]
```
Public Event OnProgress As OnProgressEventHandler
```

[C#]
```
public event OnProgressEventHandler OnProgress;
```

Event Data

The event handler receives an argument of type InternetMail.ProgressEventArgs containing data related to this event. The following **InternetMail.ProgressEventArgs** properties provide information specific to this event.

Property	Description
Message	Gets the message number.
MessageCopied	Gets a value which specifies the number of bytes of data that has been read or written.
MessageSize	Gets a value which specifies the total number of bytes in the data stream.
Percent	Gets a value which specifies the percentage of data that has been read or written.

Remarks

The **OnProgress** event occurs as a data stream is being read or written to the client. If large amounts of data are being read or written, this event can be used to update a progress bar or other user-interface component to provide the user with some visual feedback on the progress of the operation.

This event is only generated when the **ReadStream**, **WriteStream** or **StoreStream** methods are called.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.ProgressEventArgs Class

Provides data for the OnProgress event.

For a list of all members of this type, see InternetMail.ProgressEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.InternetMail.ProgressEventArgs

[Visual Basic]

```
Public Class InternetMail.ProgressEventArgs
    Inherits EventArgs
```

[C#]

```
public class InternetMail.ProgressEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ProgressEventArgs specifies the number of bytes exchanged with the mail server. This event occurs when retrieving a message from the server, and when submitting a message to the server for delivery.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

InternetMail.ProgressEventArgs Members | SocketTools Namespace

InternetMail.ProgressEventArgs Members

InternetMail.ProgressEventArgs overview

Public Instance Constructors

InternetMail.ProgressEventArgs Constructor	Initializes a new instance of the InternetMail.ProgressEventArgs class.

Public Instance Properties

Message	Gets the message number.
MessageCopied	Gets a value which specifies the number of bytes of data that has been read or written.
MessageSize	Gets a value which specifies the total number of bytes in the data stream.
Percent	Gets a value which specifies the percentage of data that has been read or written.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetMail.ProgressEventArgs Class | SocketTools Namespace

InternetMail.ProgressEventArgs Constructor

Initializes a new instance of the InternetMail.ProgressEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public InternetMail.ProgressEventArgs();
```

See Also

InternetMail.ProgressEventArgs Class | SocketTools Namespace

InternetMail.ProgressEventArgs Properties

The properties of the **InternetMail.ProgressEventArgs** class are listed below. For a complete list of **InternetMail.ProgressEventArgs** class members, see the InternetMail.ProgressEventArgs Members topic.

Public Instance Properties

Message	Gets the message number.
MessageCopied	Gets a value which specifies the number of bytes of data that has been read or written.
MessageSize	Gets a value which specifies the total number of bytes in the data stream.
Percent	Gets a value which specifies the percentage of data that has been read or written.

See Also

InternetMail.ProgressEventArgs Class | SocketTools Namespace

InternetMail.ProgressEventArgs.Message Property

Gets the message number.

[Visual Basic]
```
Public ReadOnly Property Message As Integer
```

[C#]
```
public int Message {get;}
```

Property Value

An integer value which specifies the message number.

Remarks

The **Message** property specifies the message number for the current message that is being downloaded from the mail server to the local host. If the **OnProgress** event occurs while message data is being submitted to the server, this property will return a value of zero.

See Also

InternetMail.ProgressEventArgs Class | SocketTools Namespace

InternetMail.ProgressEventArgs.MessageCopied Property

Gets a value which specifies the number of bytes of data that has been read or written.

[Visual Basic]

```
Public ReadOnly Property MessageCopied As Integer
```

[C#]

```
public int MessageCopied {get;}
```

Property Value

An integer value which specifies the number of bytes of data.

Remarks

The **MessageCopied** property specifies the number of bytes that have been read from the client and stored in the local stream buffer, or written from the stream buffer to the client.

See Also

InternetMail.ProgressEventArgs Class | SocketTools Namespace | MessageSize Property | Percent Property

InternetMail.ProgressEventArgs.MessageSize Property

Gets a value which specifies the total number of bytes in the data stream.

[Visual Basic]

```
Public ReadOnly Property MessageSize As Integer
```

[C#]

```
public int MessageSize {get;}
```

Property Value

An integer value which specifies the number of bytes of data.

Remarks

The **MessageSize** property specifies the total amount of data being read from the client and stored in the data stream, or written from the data stream to the client. If the amount of data was unknown or unspecified at the time the method call was made, then this value will always be the same as the **MessageCopied** property.

See Also

InternetMail.ProgressEventArgs Class | SocketTools Namespace | MessageCopied Property | Percent Property

InternetMail.ProgressEventArgs.Percent Property

Gets a value which specifies the percentage of data that has been read or written.

[Visual Basic]

```
Public ReadOnly Property Percent As Integer
```

[C#]

```
public int Percent {get;}
```

Property Value

An integer value which specifies a percentage.

Remarks

The **Percent** property specifies the percentage of data that has been transmitted, expressed as an integer value between 0 and 100, inclusive. If the maximum size of the data stream was not specified by the caller, this value will always be 100.

See Also

InternetMail.ProgressEventArgs Class | SocketTools Namespace | MessageCopied Property | MessageSize Property

InternetMail.OnRecipient Event

Occurs when a message is about to be submitted for delivery.

[Visual Basic]
```
Public Event OnRecipient As OnRecipientEventHandler
```

[C#]
```
public event OnRecipientEventHandler OnRecipient;
```

Event Data

The event handler receives an argument of type InternetMail.RecipientEventArgs containing data related to this event. The following **InternetMail.RecipientEventArgs** property provides information specific to this event.

Property	Description
Address	Gets the email address of the message recipient.

Remarks

The **OnRecipient** event is generated when a message is about to be submitted to the mail server for delivery. If multiple recipients have been specified for the message, this event will fire for each recipient. To prevent the message from being delivered to the specified recipient, call the **Cancel** method.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.RecipientEventArgs Class

Provides data for the OnRecipient event.

For a list of all members of this type, see InternetMail.RecipientEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.InternetMail.RecipientEventArgs

[Visual Basic]

```
Public Class InternetMail.RecipientEventArgs
    Inherits EventArgs
```

[C#]

```
public class InternetMail.RecipientEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

RecipientEventArgs specifies recipient of a message that is about to be submitted to the mail server for delivery.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

InternetMail.RecipientEventArgs Members | SocketTools Namespace

InternetMail.RecipientEventArgs Members

InternetMail.RecipientEventArgs overview

Public Instance Constructors

InternetMail.RecipientEventArgs Constructor	Initializes a new instance of the InternetMail.RecipientEventArgs class.

Public Instance Properties

Address	Gets the email address of the message recipient.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetMail.RecipientEventArgs Class | SocketTools Namespace

InternetMail.RecipientEventArgs Constructor

Initializes a new instance of the InternetMail.RecipientEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public InternetMail.RecipientEventArgs();
```

See Also

InternetMail.RecipientEventArgs Class | SocketTools Namespace

InternetMail.RecipientEventArgs Properties

The properties of the **InternetMail.RecipientEventArgs** class are listed below. For a complete list of **InternetMail.RecipientEventArgs** class members, see the InternetMail.RecipientEventArgs Members topic.

Public Instance Properties

Address	Gets the email address of the message recipient.

See Also

InternetMail.RecipientEventArgs Class | SocketTools Namespace

InternetMail.RecipientEventArgs.Address Property

Gets the email address of the message recipient.

[Visual Basic]

```
Public ReadOnly Property Address As String
```

[C#]

```
public string Address {get;}
```

Property Value

A string value which specifies the recipient of the email message.

Remarks

The **Address** property returns the email address of the recipient for the message that is about to be submitted to the mail server for delivery.

See Also

InternetMail.RecipientEventArgs Class | SocketTools Namespace

InternetMail.OnTimeout Event

Occurs when a blocking operation fails to complete before the timeout period elapses.

[Visual Basic]

```
Public Event OnTimeout As EventHandler
```

[C#]

```
public event EventHandler OnTimeout;
```

Remarks

The **OnTimeout** event occurs when a blocking operation, such as sending or receiving data on the client, fails to complete before the specified timeout period elapses. The timeout period for a blocking operation can be adjusted by setting the **Timeout** property.

See Also

InternetMail Class | SocketTools Namespace

InternetMail.UpdateEventArgs Class

Provides data for the OnUpdate event.

For a list of all members of this type, see InternetMail.UpdateEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.InternetMail.UpdateEventArgs

[Visual Basic]

```
Public Class InternetMail.UpdateEventArgs
    Inherits EventArgs
```

[C#]

```
public class InternetMail.UpdateEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

InternetMail.UpdateEventArgs Members | SocketTools Namespace

InternetMail.UpdateEventArgs Members

InternetMail.UpdateEventArgs overview

Public Instance Constructors

InternetMail.UpdateEventArgs Constructor	Initializes a new instance of the InternetMail.UpdateEventArgs class.

Public Instance Properties

Message	Gets the message number.
UpdateType	Gets the type of update notification that has been sent by the server.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetMail.UpdateEventArgs Class | SocketTools Namespace

InternetMail.UpdateEventArgs Constructor

Initializes a new instance of the InternetMail.UpdateEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public InternetMail.UpdateEventArgs();
```

See Also

InternetMail.UpdateEventArgs Class | SocketTools Namespace

InternetMail.UpdateEventArgs Properties

The properties of the **InternetMail.UpdateEventArgs** class are listed below. For a complete list of **InternetMail.UpdateEventArgs** class members, see the InternetMail.UpdateEventArgs Members topic.

Public Instance Properties

Message	Gets the message number.
UpdateType	Gets the type of update notification that has been sent by the server.

See Also

InternetMail.UpdateEventArgs Class | SocketTools Namespace

InternetMail.UpdateEventArgs.Message Property

Gets the message number.

[Visual Basic]

```
Public ReadOnly Property Message As Integer
```

[C#]

```
public int Message {get;}
```

Property Value

An integer value which specifies the message number.

Remarks

The **Message** property specifies the message number for the message that has been added to or expunged from the current mailbox. A value of zero indicates that the update notification does not reference a specific message in the mailbox.

See Also

InternetMail.UpdateEventArgs Class | SocketTools Namespace

InternetMail.UpdateEventArgs.UpdateType Property

Gets the type of update notification that has been sent by the server.

[Visual Basic]

```
Public ReadOnly Property UpdateType As IdleUpdate
```

[C#]

```
public InternetMail.IdleUpdate UpdateType {get;}
```

Property Value

One or more of the IdleUpdate enumeration flags.

See Also

InternetMail.UpdateEventArgs Class | SocketTools Namespace

InternetMail.OnUpdate Event

Occurs when the server sends a mailbox update notification to the client.

[Visual Basic]

```
Public Event OnUpdate As OnUpdateEventHandler
```

[C#]

```
public event OnUpdateEventHandler OnUpdate;
```

Event Data

The event handler receives an argument of type InternetMail.UpdateEventArgs containing data related to this event. The following **InternetMail.UpdateEventArgs** properties provide information specific to this event.

Property	Description
Message	Gets the message number.
UpdateType	Gets the type of update notification that has been sent by the server.

Remarks

This event is only generated when the **Idle** method has been used to enable mailbox status monitoring.

See Also

InternetMail Class | SocketTools Namespace | Idle Method | OnUpdateEventHandler Delegate

InternetMail.OnDeliveredEventHandler Delegate

Represents the method that will handle the OnDelivered event.

[Visual Basic]

```
Public Delegate Sub InternetMail.OnDeliveredEventHandler( _
   ByVal sender As Object, _
   ByVal e As DeliveredEventArgs _
)
```

[C#]

```
public delegate void InternetMail.OnDeliveredEventHandler(
      object sender,
      DeliveredEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An DeliveredEventArgs that contains the event data.

Remarks

When you create an **OnDeliveredEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnDeliveredEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace

InternetMail.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub InternetMail.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void InternetMail.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ErrorEventArgs that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace

InternetMail.OnProgressEventHandler Delegate

Represents the method that will handle the OnProgress event.

```
[Visual Basic]
Public Delegate Sub InternetMail.OnProgressEventHandler( _
   ByVal sender As Object, _
   ByVal e As ProgressEventArgs _
)
```

```
[C#]
public delegate void InternetMail.OnProgressEventHandler(
      object sender,
      ProgressEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A ProgressEventArgs that contains the event data.

Remarks

When you create an **OnProgressEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnProgressEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace

InternetMail.OnRecipientEventHandler Delegate

Represents the method that will handle the OnRecipient event.

[Visual Basic]

```
Public Delegate Sub InternetMail.OnRecipientEventHandler( _
   ByVal sender As Object, _
   ByVal e As RecipientEventArgs _
)
```

[C#]

```
public delegate void InternetMail.OnRecipientEventHandler(
      object sender,
      RecipientEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An RecipientEventArgs that contains the event data.

Remarks

When you create an **OnRecipientEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnRecipientEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace

InternetMail.OnUpdateEventHandler Delegate

Represents the method that will handle the OnUpdate event.

[Visual Basic]

```
Public Delegate Sub InternetMail.OnUpdateEventHandler( _
   ByVal sender As Object, _
   ByVal e As UpdateEventArgs _
)
```

[C#]

```
public delegate void InternetMail.OnUpdateEventHandler(
      object sender,
      UpdateEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An UpdateEventArgs that contains the event data.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace | UpdateEventArgs Class

InternetMail.ErrorCode Enumeration

Specifies the error codes returned by the InternetMail class.

[Visual Basic]
```
Public Enum InternetMail.ErrorCode
```

[C#]
```
public enum InternetMail.ErrorCode
```

Remarks

The InternetMail class uses the **ErrorCode** enumeration to specify what error has occurred when a network operation fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous network operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote

	host.
errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.

errorSecurityCertificate	Unable to validate the certificate chain for this session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.

errorProxyAuthenticationRequired	Proxy authentication required.
errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.

errorUnsupportedMediaType	Request specified an unsupported media type.
errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.

errorNoMoreMailboxes	No more mailboxes exist on this server.
errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.

errorInvalidThreadHandle	The specified thread handle is no longer valid.
errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been

	defined.
errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or

	unrecognized.
errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

InternetMail.IdleOptions Enumeration

Specifies the idle monitoring options that the InternetMail class supports.

[Visual Basic]
```
Public Enum InternetMail.IdleOptions
```

[C#]
```
public enum InternetMail.IdleOptions
```

Remarks

The **Idle** method can operate in two modes, based on the options specified by the caller. If the option **imapIdleNoWait** is specified, the method begins monitoring the client session asynchronously and returns control immediately to the caller. If the server sends a update notification to the client, the **OnUpdate** event will fire with information about the status change. If the option **imapIdleWait** is specified, the method will block waiting for the server to send a notification message to the client. The method will return when either a message is received or the timeout period is exceeded.

Members

Member Name	Description
idleNoWait	The **Idle** method should return immediately after idle processing has been enabled. When this option is used, the application may continue to perform other functions while the client session is monitored for status updates sent by the server. The client will continue to monitor status changes until an IMAP command issued or the client disconnects from the server. This is the default option.
idleWait	The **Idle** method should wait until the server sends a status update, or until the timeout period is reached. The client will stop monitoring status changes when the function returns. If this option is used in a single-threaded application, normal message processing can be impeded, causing the application to appear non-responsive until the timeout period is reached. It is strongly recommended that single-threaded applications with a user interface specify the **imapIdleNoWait** option instead.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace | Idle Method (SocketTools.InternetMail) | OnUpdate Event (SocketTools.InternetMail)

InternetMail.IdleUpdate Enumeration

Specifies the types of update notification messages that the InternetMail class supports.

[Visual Basic]

```
Public Enum InternetMail.IdleUpdate
```

[C#]

```
public enum InternetMail.IdleUpdate
```

Members

Member Name	Description
updateUnknown	The server has sent an unrecognized notification message. This does not necessarily reflect an error condition, as some servers may send additional notification messages beyond the standard EXISTS, EXPUNGE and RECENT messages. Most applications should ignore this type of notification.
updateMessage	The server has sent notification message to the client indicating that a new message has arrived. Typically this update notification occurs shortly after the new message has been stored in the current mailbox.
updateExpunge	The server has sent a notification message to the client indicating that a message has been removed from the current mailbox. It is recommended that the application re-examine the mailbox when this notification is received. Typically this notification is only sent periodically by the server, and may not be sent immediately after a message has been expunged from the mailbox.
updateMailbox	The server has sent notification message to the client indicating that the state of the mailbox has changed. This message is sent periodically by the server and may not be sent immediately after a new message arrives or a message is flagged as unread. It is recommended that the application re-examine the mailbox when this notification is received.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace | OnUpdate

InternetMail.ImapFlags Enumeration

Specifies the mailbox and message flags that the InternetMail class supports when connected to a mail server using the IMAP4 protocol.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum InternetMail.ImapFlags
```

[C#]

```
[Flags]
public enum InternetMail.ImapFlags
```

Members

Member Name	Description	Value
flagNone	No flags have been set for the current message or mailbox.	0
flagAnswered	The message has been answered.	1
flagDraft	The message is a draft copy and has not been delivered.	2
flagUrgent	The message has been flagged for urgent or special attention.	4
flagSeen	The message has been read.	8
flagRecent	The message has been added to the mailbox recent.	256
flagDeleted	The message has been marked for deletion.	512
flagNoInferiors	The mailbox does not contain any child mailboxes. In the IMAP protocol, these are referred to as inferior hierarchical mailbox names.	65536
flagNoSelect	The mailbox cannot be selected or examined. This flag is typically used by servers to indicate that the mailbox name refers to a directory on the server, not an actual mailbox.	131072
flagMarked	The mailbox is marked as being of interest to a client. If this flag is used, it typically means that the mailbox contains messages. An application should not depend on this flag being present for any given mailbox. Some IMAP servers do not support marked or unmarked flags for mailboxes.	262144
flagUnmarked	The mailbox is marked as not being of	524288

	interest to a client. If this flag is used, it typically means that the mailbox does not contain any messages. An application should not depend on this flag being present for any given mailbox. Some IMAP servers do not support marked or unmarked flags for mailboxes.	

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace

InternetMail.ImapResult Enumeration

Specifies the result codes that the InternetMail class supports.

[Visual Basic]
```
Public Enum InternetMail.ImapResult
```

[C#]
```
public enum InternetMail.ImapResult
```

Members

Member Name	Description
resultUnknown	An unknown result code was returned by the server.
resultOk	The previous command completed successfully. The result string contains information about the results of the command.
resultNo	The previous command could not be completed. The result string contains information about why the command failed.
resultBad	The previous command could not be completed, the command may be invalid or not supported on the server. The result string contains information about why the command failed.
resultContinue	The command has executed and is waiting for additional data from the client.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace

InternetMail.ImapSections Enumeration

Specifies the message sections that the InternetMail class supports when connected to a mail server using the IMAP4 protocol.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum InternetMail.ImapSections
```

```
[C#]
[Flags]
public enum InternetMail.ImapSections
```

Members

Member Name	Description	Value
sectionDefault	All headers and the complete body of the specified message or message part are to be retrieved. The client application is responsible for parsing the header block. If the message is a MIME multipart message and the complete message is returned, the application is responsible for parsing the individual message parts if necessary.	0
sectionHeader	All headers for the specified message or message part are to be retrieved. The client application is responsible for parsing the header block.	1
sectionMimeHeader	The MIME headers for the specified message or message are to be retrieved. Only those header fields which are used in MIME messages, such as Content-Type will be returned to the client. This is typically useful when processing the header for a multipart message which contains file attachments. The client application is responsible for parsing the header block.	2
sectionBody	The body of the specified message or message part will be retrieved. For a MIME formatted message, this may include data that is uuencoded or base64 encoded. The application is responsible for decoding this data.	4
sectionPreview	The message header or body is being previewed and should not be marked as read by the server. This prevents the	4096

	message from having the IMAP_FLAG_SEEN flag from being automatically set when the message data is retrieved.	

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace

InternetMail.InternetMailOptions Enumeration

Specifies the options that the InternetMail class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum InternetMail.InternetMailOptions
```

[C#]

```
[Flags]
public enum InternetMail.InternetMailOptions
```

Remarks

The InternetMail class uses the **InternetMailOptions** enumeration to specify one or more options to be used when establishing a connection with a remote host. Multiple options may be specified if necessary.

Members

Member Name	Description	Value
optionNone	No option specified.	0
optionAuthLogin	This option specifies the client should attempt to authenticate with the mail server when submitting messages for delivery. This option is typically required when relaying messages through a server to a recipient in a different domain.	1
optionAllHeaders	Preserves all headers in the message when it is exported, including the Received and Return-Path headers which are normally excluded.	256
optionKeepOrder	Preserves the order of the headers when it is exported; by default, some headers may be re-ordered.	512
optionSecure	This option specifies the client should attempt to establish a secure connection with the server. Note that the server must support secure connections using either the SSL or TLS protocol.	4096
optionExplicitSSL	This option specifies the client should attempt to establish a secure explicit SSL session. The initial connection to the server is not encrypted, and the client will attempt to negotiate a secure connection by sending a command to the server. Some servers may require this option when connecting to the	4096

	server on ports other than the default secure port.	
optionImplicitSSL	This option specifies the client should attempt to establish a secure implicit SSL session. The SSL handshake is initiated immediately after the connection to the server has been established.	8192
optionSecureFallback	This option specifies the client should permit the use of less secure cipher suites for compatibility with legacy servers. If this option is specified, the client will permit connections using TLS 1.0 and cipher suites that use RC4, MD5 and SHA1.	32768
optionFreeThread	This option specifies that class methods may be called from any thread, and not only the thread that established the connection. Using this option disables certain internal safety checks that are made by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads.	524288
optionNotify	Notify the sender of the delivery status of the message, if the server supports delivery status notification. This option is a combination of the optionNotifySuccess, optionNotifyFailure, optionNotifyDelay and optionReturnHeaders options.	983040
optionNotifySuccess	If the mail server supports delivery status notification, this causes a message to be returned to the sender once it has been successfully delivered.	65536
optionNotifyFailure	If the mail server supports delivery status notification, this causes a message to be returned to the sender if it could not be delivered.	131072
optionNotifyDelay	If the mail server supports delivery status notification, this causes a message to be returned to the sender if delivery has been delayed.	262144
optionReturnHeaders	If the mail server supports delivery status notification, this causes a message to be returned which contains the headers of the message that was sent.	524288

optionReturnMessage	If the mail server supports delivery status notification, this causes a message to be returned which contains the complete message that was sent.	1048576

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace

InternetMail.MailServerType Enumeration

Specifies the type of mail server that the client is connected to.

[Visual Basic]
```
Public Enum InternetMail.MailServerType
```

[C#]
```
public enum InternetMail.MailServerType
```

Members

Member Name	Description
serverUnknown	The server type has not been explicitly set. The server type will be automatically determined by the value of the service port number specified by the value of the **ServerPort** property.
serverPop3	The Post Office Protocol is used when establishing a connection to the mail server.
serverImap4	The Internet Message Access Protocol is used when establishing a connection to the mail server.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace | ServerType Property (SocketTools.InternetMail)

InternetMail.MimeAttachment Enumeration

Specifies the file attachment options supported by the InternetMail class.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum InternetMail.MimeAttachment
```

```
[C#]
[Flags]
public enum InternetMail.MimeAttachment
```

Remarks

The **attachAlternative** and **attachInline** values can be combined with one of the attachment options using a bitwise OR operator.

Members

Member Name	Description	Value
attachDefault	The file attachment encoding is based on the file content type. Text files are not encoded, and binary files are encoded using the standard base64 encoding algorithm. This is the default option for file attachments.	0
attachBase64	The file attachment is always encoded using the standard base64 algorithm, even if the attached file is a plain text file.	1
attachUucode	The file attachment is always encoded using the uuencode algorithm, even if the attached file is a plain text file.	2
attachQuoted	The file attachment is always encoded using the quoted-printable algorithm, even if the attached file is a plain text file.	3
attachAlternative	The attached data is an alternative format for the contents of the message. This can only be used with textual data.	65536
attachInline	The attached data will be displayed inline with the contents of the message. This is typically used with images that are to be displayed along with the message text.	131072

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace

InternetMail.MimeCharacterSet Enumeration

Specifies the character sets recognized by the InternetMail class.

[Visual Basic]

```
Public Enum InternetMail.MimeCharacterSet
```

[C#]

```
public enum InternetMail.MimeCharacterSet
```

Members

Member Name	Description
charsetUnknown	The character set is unknown.
charsetDefault	The default character set. This is the same as specifying the character set **charsetUTF8**.
charsetUSASCII	A character set using US-ASCII which defines 7-bit printable characters with values ranging from 20h to 7Eh. An application that uses this character set has the broadest compatibility with most mail servers (MTAs) because it does not require the server to handle 8-bit characters correctly when the message is delivered. This is the most commonly used character set for plain text email messages in the English language and is the default character set used by the class.
charsetISO8859_1	An 8-bit character set for most western European languages such as English, French, Spanish and German. This character set is also commonly referred to as Latin1. The Windows code page for this character set is 28591, however Windows code page 1252 (Windows-1252) is typically used to represent this character set in most applications.
charsetISO8859_2	An 8-bit character set for most central and eastern European languages such as Czech, Hungarian, Polish and Romanian. This character set is also commonly referred to as Latin2. This character set is similar to Windows code page 1250, however the characters are arranged differently.
charsetISO8859_3	A character set for southern European languages such as Maltese and Esperanto. This character set was also used with the Turkish language, but it was superseded by ISO 8859-9 which is the preferred character set for Turkish. This character set is not widely used in mail messages and it is recommended that you use UTF-8 instead.
charsetISO8859_4	A character set for northern European languages such as Latvian, Lithuanian and Greenlandic. This character set is not widely used in mail messages

	and it is recommended that you use UTF-8 instead.
charsetISO8859_5	An 8-bit character set for Cyrillic languages such as Russian, Bulgarian and Serbian. The Windows code page for this character set is 28595. This character set is not widely used and it is recommended that you use UTF-8 instead.
charsetISO8859_6	An 8-bit character set for Arabic languages. Note that the application is responsible for displaying text that uses this character set. In particular, any display engine needs to be able to handle the reverse writing direction and analyze the context of the message to correctly combine the glyphs. This character set is not widely used and it is recommended that you use UTF-8 instead.
charsetISO8859_7	An 8-bit character set for the Greek language. This character set is also commonly referred to as Latin/Greek. The Windows code page for this character set is 28597.
charsetISO8859_8	An 8-bit character set for the Hebrew language. Note that similar to Arabic, Hebrew uses a reverse writing direction. An application which displays this character should be capable of processing bi-directional text where a single message may include both right-to-left and left-to-right languages, such as Hebrew and English. The Windows code page for this character set is 28598.
charsetISO8859_9	An 8-bit character set for the Turkish language. This character set is also commonly referred to as Latin5. The Windows code page for this character set is 28599.
charsetISO8859_10	A character set for the Danish, Icelandic, Norwegian and Swedish languages. This character set is also commonly referred to as Latin-6 and is similar to ISO 8859-4.
charsetISO8859_13	A character set for Baltic languages. This character set is also commonly referred to as Latin-7. This character set is similar to ISO 8859-4, except it adds certain Polish characters and does not support Nordic languages.
charsetISO8859_14	A character set for Gaelic languages such as Irish, Manx and Scottish Gaelic. This character set is also commonly referred to as Latin-8. This character set replaced ISO 8859-12 which was never fully implemented.
charsetISO8859_15	A character set for western European languages. This character set is also commonly referred to as Latin-9 and is nearly identical to ISO8859-1 except

	that it replaces lesser-used symbols with the Euro sign and some letters.
charsetISO2022_JP	A multi-byte character encoding for Japanese that is widely used with mail messages. This is a 7-bit encoding where all characters start with ASCII and uses escape sequences to switch to the double-byte character sets.
charsetISO2022_KR	A multi-byte character encoding for Korean which encodes both ASCII and Korean double-byte characters. This is a 7-bit encoding which uses the shift in and shift out control characters to switch to the double-byte character set.
charsetISO2022_CN	A multi-byte character encoding for Simplified Chinese which encodes both ASCII and Chinese double-byte characters. This is a 7-bit encoding which uses the shift in and shift out control characters to switch to the double-byte character set.
charsetKOI8R	A character set for Russian using the Cyrillic alphabet. This character set also covers the Bulgarian language. Most mail messages in the Russian language use this character set or UTF-8 instead of ISO 8859-5, which was never widely adopted.
charsetKOI8U	A character set for Ukrainian using the Cyrillic alphabet. This character set is similar to the KOI8-R character set, but replaces certain symbols with Ukrainian letters. Most mail messages in the Ukrainian language use this character set or UTF-8 instead of ISO 8859-5, which was never widely adopted.
charsetGB2312	A multi-byte character encoding which can represent ASCII and simplified Chinese characters. It has been superseded by GB18030, however it remains widely used in China.
charsetGB18030	A Unicode transformation format which can represent all Unicode code points and supports both simplified and traditional Chinese characters. It is backwards compatible with GB2312 and supersedes that character set.
charsetBIG5	A multi-byte character set that supports both ASCII characters and traditional Chinese characters. It is widely used in Taiwan, Hong Kong and Macau. It is no longer commonly used in China, which has developed GB18030 as a standard encoding. Note that Microsoft's implementation of Big5 on Windows does not support all of the extensions and is missing certain

	code points.
charsetUTF7	A 7-bit Unicode Transformation Format that uses variable-length character encoding to represent Unicode text as a stream of ASCII characters that are safe to transport between mail servers that only support 7-bit printable characters. It is primarily used as an alternative to UTF-8 which requires that the mail server support 8-bit text or use quoted-printable encoding.
charsetUTF8	An 8-bit Unicode Transformation Format that uses multibyte character sequences to represent Unicode text. It is backwards compatible with the ASCII character set, however because it uses 8-bit text, it should be encoded using either quoted-printable or base64 encoding to ensure that mail servers that do not support 8-bit characters.
charsetUTF16	A 16-bit Unicode Transformation Format that uses two bytes to represent each Unicode character. Messages that use UTF-16 are commonly encoded using the base64 algorithm. It is recommended that most applications use the UTF-8 character set, which is capable of representing all Unicode characters.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace

InternetMail.MimeContent Enumeration

Specifies the content types supported by the InternetMail class.

[Visual Basic]

```
Public Enum InternetMail.MimeContent
```

[C#]

```
public enum InternetMail.MimeContent
```

Members

Member Name	Description
contentUnknown	The content type is unknown. This value may be returned if the message handle is invalid, or if the file extension is unknown and the file could not be opened for read access.
contentDefault	The default content type. This is the same as specifying the content type **contentText**.
contentApplication	The content is application specific. Examples of this type of file would be a Microsoft Word document or an executable program. This is also the default type for files which have an unrecognized file name extension and contain binary data.
contentAudio	The content is audio data in one of several standard formats. Examples of this type of file would be a Windows (.wav) file or MPEG3 (.mp3) file.
contentImage	The content is an image data in one of several standard formats. Examples of this type of file would be a GIF or JPEG image file.
contentMessage	The content is an email message encapsulated within the current message.
contentMultipart	The content is a multipart MIME email message which contains additional message parts. For example, an email message which contains both a text message and file attachment would be identified as a multipart message.
contentText	The content is textual data. This is also the default type for files which have an unrecognized file name extension and contain only printable text.
contentVideo	The content is video data in one of several standard formats. Examples of this type of file would be a Windows (.avi) or Quicktime (.mov) video file.
contentWideText	The content is Unicode text. This is also the default type for files which have an unrecognized file

	name extension. The content must be prefixed with a byte order mark (BOM) to be recognized as Unicode text.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace

InternetMail.MimeEncoding Enumeration

Specifies the encoding types supported by the InternetMail class.

[Visual Basic]
```
Public Enum InternetMail.MimeEncoding
```

[C#]
```
public enum InternetMail.MimeEncoding
```

Members

Member Name	Description
encodingUnknown	The encoding type is unknown.
encodingDefault	The encoding type is based on the content type. Text data is not encoded, and binary data is encoded using the standard base64 encoding algorithm. This is the default option for file attachments.
encoding7Bit	Each character is encoded in one or more bytes, with each byte being 8 bits long, with the most significant bit cleared. This encoding is most commonly used with plain text using the US-ASCII character set, where each character is represented by a single byte in the range of 20h to 7Eh.
encoding8Bit	Each character is encoded in one or more bytes, with each byte being 8 bits long and all bits are used. 8-bit encoding is typically used with multibyte character sets and is the default encoding used with Unicode text.
encodingBinary	The data contains unmodified 8-bit data. This encoding type is rarely specified for email messages because not all mail servers are capable of transmitting 8-bit data. In most cases, messages which contain binary data are encoded using the base64 algorithm.
encodingQuoted	The data is encoded using quoted-printable encoding. Printable ASCII characters are left as-is, with non-printable characters encoded as their hexadecimal value. Quoted-printable encoding is commonly used with HTML formatted email messages.
encodingBase64	The data is encoded using the standard base64 algorithm. This is the most common encoding method for binary data in an email message.
encodingUucode	The data is encoded using the uuencode algorithm. This encoding method is common for binary attachments to news articles, but is rarely

	used with email messages.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace

InternetMail.MimeExportOptions Enumeration

Specifies the export options that the InternetMail class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum InternetMail.MimeExportOptions
```

[C#]

```
[Flags]
public enum InternetMail.MimeExportOptions
```

Members

Member Name	Description	Value
exportDefault	The default export options. The headers for the message are written out in a specific consistent order, with custom headers written to the end of the header block regardless of the order in which they were set or imported from another message. If the message contains Bcc, Received, Return-Path, Status or X400-Received header fields, they will not be exported.	0
exportAllHeaders	All headers, including the Bcc, Received, Return-Path, Status and X400-Received header fields will be exported. Normally these headers are not exported because they are only used by the mail transport system. This option can be useful when exporting a message to be stored on the local system, but should not be used when exporting a message to be delivered to another user.	1
exportKeepOrder	The original order in which the message header fields were set or imported are preserved when the message is exported.	2

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace

InternetMail.PopAuthentication Enumeration

Specifies the authentication methods supported by the InternetMail class when connected to a mail server using the POP3 protocol.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum InternetMail.PopAuthentication
```

[C#]

```
[Flags]
public enum InternetMail.PopAuthentication
```

Members

Member Name	Description	Value
authDefault	The default authentication type. This is the same as specifying the **authPass** authentication type.	0
authPassword	Standard cleartext username and password is sent to the server. This authentication method is supported by all servers.	0
authApop	The APOP authentication method which uses an MD5 digest of the password. This method is not supported by all servers and should only be specified if required by the server.	1

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace

InternetMail.SecureCipherAlgorithm Enumeration

Specifies the encryption algorithms that the InternetMail class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum InternetMail.SecureCipherAlgorithm
```

[C#]
```
[Flags]
public enum InternetMail.SecureCipherAlgorithm
```

Remarks

The InternetMail class uses the **SecureCipherAlgorithm** enumeration to identify which encryption algorithm was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
cipherNone	No cipher has been selected. A secure connection has not been established with the remote host.	0
cipherRC2	The RC2 block cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.	1
cipherRC4	The RC4 stream cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.	2
cipherRC5	The RC5 block cipher was selected. This is a variable key length cipher which supports keys up to 2040 bits, in 8-bit increments.	4
cipherDES	The DES (Data Encryption Standard) block cipher was selected. This is a fixed key length cipher using 56-bit keys.	8
cipherDES3	The Triple DES block cipher was selected. This cipher encrypts the data three times using different keys, effectively using a 168-bit key length.	16
cipherDESX	A variant of the DES block cipher which XORs an extra 64-bits of the key before and after the plaintext has been encrypted, increasing the key size to 184 bits.	32
cipherAES	The Advanced Encryption Standard	64

	cipher (also known as the Rijndael cipher) is a fixed block size cipher which use a key size of 128, 192 or 256 bits. This cipher is supported on Windows XP SP3 SP3 and later versions of the operating system.	
cipherSkipjack	The Skipjack block cipher was selected. This is a fixed key length cipher, using 80-bit keys.	128

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace

InternetMail.SecureHashAlgorithm Enumeration

Specifies the hash algorithms that the InternetMail class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum InternetMail.SecureHashAlgorithm
```

```
[C#]
[Flags]
public enum InternetMail.SecureHashAlgorithm
```

Remarks

The InternetMail class uses the **SecureHashAlgorithm** enumeration to identify the message digest (hash) algorithm that was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
hashNone	No hash algorithm has been selected. This is not a secure connection with the server.	0
hashMD5	The MD5 algorithm was selected. This algorithm produces a 128-bit message digest. This algorithm is no longer considered to be cryptographically secure.	1
hashSHA	The SHA-1 algorithm was selected. This algorithm produces a 160-bit message digest. This algorithm is no longer considered to be cryptographically secure.	2
hashSHA256	The SHA-256 algorithm was selected. This algorithm produces a 256-bit message digest.	4
hashSHA384	The SHA-384 algorithm was selected. This algorithm produces a 384-bit message digest.	8
hashSHA512	The SHA-512 algorithm was selected. This algorithm produces a 512-bit message digest.	16

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace

InternetMail.SecureKeyAlgorithm Enumeration

Specifies the key exchange algorithms that the InternetMail class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum InternetMail.SecureKeyAlgorithm
```

[C#]

```
[Flags]
public enum InternetMail.SecureKeyAlgorithm
```

Remarks

The InternetMail class uses the **SecureKeyAlgorithm** enumeration to identify the key exchange algorithm that was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
keyExchangeNone	No key exchange algorithm has been selected. This is not a secure connection with the server.	0
keyExchangeRSA	The RSA public key exchange algorithm has been selected.	1
keyExchangeKEA	The KEA public key exchange algorithm has been selected. This is an improved version of the Diffie-Hellman public key algorithm.	2
keyExchangeDH	The Diffie-Hellman public key exchange algorithm has been selected.	4
keyExchangeECDH	The Elliptic Curve Diffie-Hellman key exchange algorithm was selected. This is a variant of the Diffie-Hellman algorithm which uses elliptic curve cryptography. This key exchange algorithm is only supported on Windows XP SP3 SP3 and later versions of the operating system.	8

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace

InternetMail.SecurityCertificate Enumeration

Specifies the security certificate status values that may be returned by the InternetMail class.

[Visual Basic]

```
Public Enum InternetMail.SecurityCertificate
```

[C#]

```
public enum InternetMail.SecurityCertificate
```

Remarks

The InternetMail class uses the **SecurityCertificate** enumeration to identify the current status of the certificate that was provided by the remote host when a secure connection was established.

Members

Member Name	Description
certificateNone	No certificate information is available. A secure connection was not established with the server.
certificateValid	The certificate is valid.
certificateNoMatch	The certificate is valid, however the domain name specified in the certificate does not match the domain name of the remote host. The application can examine the **CertificateSubject** property to determine the site the certificate was issued to.
certificateExpired	The certificate has expired and is no longer valid. The application can examine the **CertificateExpires** property to determine when the certificate expired.
certificateRevoked	The certificate has been revoked and is no longer valid. It is recommended that the application immediately terminate the connection if this status is returned.
certificateUntrusted	The certificate has not been issued by a trusted authority, or the certificate is not trusted on the local host. It is recommended that the application immediately terminate the connection if this status is returned.
certificateInvalid	The certificate is invalid. This typically indicates that the internal structure of the certificate is damaged. It is recommended that the application immediately terminate the connection if this status is returned.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace

InternetMail.SecurityProtocols Enumeration

Specifies the security protocols that the InternetMail class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum InternetMail.SecurityProtocols
```

[C#]

```
[Flags]
public enum InternetMail.SecurityProtocols
```

Remarks

The InternetMail class uses the **SecurityProtocols** enumeration to specify one or more security protocols to be used when establishing a connection with a remote host. Multiple protocols may be specified if necessary and the actual protocol used will be negotiated with the remote host. It is recommended that most applications use **protocolDefault** when creating a secure connection.

Members

Member Name	Description	Value
protocolNone	No security protocol will be used, a secure connection will not be established.	0
protocolSSL2	The SSL 2.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections.	1
protocolSSL3	The SSL 3.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections. In most cases, this protocol is only selected if TLS is not supported by the server.	2
protocolTLS10	The TLS 1.0 protocol has been selected. This version of the protocol is commonly used by older servers and is the only version of TLS supported on Windows platforms prior to Windows 7 SP1 and Windows Server 2008 R2.	4
protocolTLS11	The TLS 1.1 protocol has been selected. This version of TLS is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of the	8

	operating system.	
protocolTLS12	The TLS 1.2 protocol has been selected. This is the default version of the protocol and is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of Windows. It is recommended that you use this version of TLS.	16
protocolSSL	Any version of the Secure Sockets Layer (SSL) protocol should be used. The actual protocol version used will be negotiated with the remote host.	3
protocolTLS	Any version of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with with preference for TLS 1.2.	28
protocolTLS1	Version 1.0, 1.1 or 1.2 of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with preference for TLS 1.2.	28
protocolDefault	The default selection of security protocols will be used when establishing a connection. The TLS 1.2, 1.1 and 1.0 protocols will be negotiated with the host, in that order of preference. This option will always request the latest version of the preferred security protocols and is the recommended value.	16
protocolUnknown	An unknown or unsupported security protocol has been specified. This value indicates an error condition.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace

InternetMail.ThreadingModel Enumeration

Specifies the threading model used by the class instance.

[Visual Basic]

```
Public Enum InternetMail.ThreadingModel
```

[C#]

```
public enum InternetMail.ThreadingModel
```

Remarks

The threading model **modelSingleThread** does not limit the application to a single thread of execution. It specifies that only a single thread may invoke methods in a class instance. When a session is established using the **Connect** method, that session is attached to the thread that created it. From that point on, until the session is terminated, only the owner may invoke methods in that instance of the class. The ownership of the class instance may be transferred from one thread to another using the **AttachThread** method.

The threading model **modelFreeThread** disables certain internal safety checks that are performed by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads. The application must ensure that no two threads will attempt to invoke a blocking method at the same time. In other words, if one thread invokes a method, the application must ensure that another thread will not attempt to invoke any other method at the same time using the same instance of the class.

Members

Member Name	Description
modelSingleThread	Methods in the class instance may only be invoked by a single thread. This threading model specifies that only the thread which established the connection should be permitted to invoke methods. This is the default threading model.
modelFreeThread	Methods in the class instance may be invoked by any thread. This threading model permits methods to be invoked across multiple threads without being explicitly attached to the object.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace

InternetMail.ThreadModelAttribute.Model Enumeration

Specifies the threading model used by the class instance.

[Visual Basic]
```
Public Enum InternetMail.ThreadModelAttribute.Model
```

[C#]
```
public enum InternetMail.ThreadModelAttribute.Model
```

Remarks

The threading model **SingleThread** does not limit the application to a single thread of execution. It specifies that only a single thread may invoke methods in a class instance. When a session is established using the **Connect** method, that session is attached to the thread that created it. From that point on, until the session is terminated, only the owner may invoke methods in that instance of the class. The ownership of the class instance may be transferred from one thread to another using the **AttachThread** method.

The threading model **FreeThread** disables certain internal safety checks that are performed by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads. The application must ensure that no two threads will attempt to invoke a blocking method at the same time. In other words, if one thread invokes a method, the application must ensure that another thread will not attempt to invoke any other method at the same time using the same instance of the class.

Members

Member Name	Description
SingleThread	Methods in the class instance may only be invoked by a single thread. This threading model specifies that only the thread which established the connection should be permitted to invoke methods. This is the default threading model.
FreeThread	Methods in the class instance may be invoked by any thread. This threading model permits methods to be invoked across multiple threads without being explicitly attached to the object.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace

InternetMail.TraceOptions Enumeration

Specifies the logging options that the InternetMail class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum InternetMail.TraceOptions
```

[C#]

```
[Flags]
public enum InternetMail.TraceOptions
```

Remarks

The InternetMail class uses the **TraceOptions** enumeration to specify what kind of debugging information is written to the trace logfile. These options are only meaningful when trace logging is enabled by setting the **Trace** property to **true**.

Members

Member Name	Description	Value
traceDefault	The default trace logging option. This is the same as specifying the **traceInfo** option.	0
traceInfo	All network function calls are written to the trace file. This is the default value.	0
traceError	Only those network function calls which fail are recorded in the trace file.	1
traceWarning	Only those network function calls which fail, or return values which indicate a warning, are recorded in the trace file.	2
traceHexDump	All network function calls are written to the trace file, plus all the data that is sent or received is displayed, in both ASCII and hexadecimal format.	4
traceProcess	All function calls in the current process are logged, rather than only those functions in the current thread. This option is useful for multithreaded applications that are using worker threads.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

SocketTools Namespace

InternetMail.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see InternetMail.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.InternetMail.RuntimeLicenseAttribute

[Visual Basic]
```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class InternetMail.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]
```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class InternetMail.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the InternetMail class.

Example

```
<Assembly: SocketTools.InternetMail.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.InternetMail.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

InternetMail.RuntimeLicenseAttribute Members | SocketTools Namespace

InternetMail.RuntimeLicenseAttribute Members

InternetMail.RuntimeLicenseAttribute overview

Public Instance Constructors

InternetMail.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetMail.RuntimeLicenseAttribute Class | SocketTools Namespace

InternetMail.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public InternetMail.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the InternetMail class.

See Also

InternetMail.RuntimeLicenseAttribute Class | SocketTools Namespace

InternetMail.RuntimeLicenseAttribute Properties

The properties of the **InternetMail.RuntimeLicenseAttribute** class are listed below. For a complete list of **InternetMail.RuntimeLicenseAttribute** class members, see the InternetMail.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

InternetMail.RuntimeLicenseAttribute Class | SocketTools Namespace

InternetMail.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

InternetMail.RuntimeLicenseAttribute Class | SocketTools Namespace

InternetMail.ThreadModelAttribute Class

Attribute that defines the threading model for the class.

For a list of all members of this type, see InternetMail.ThreadModelAttribute Members.

System.Object
  System.Attribute
    SocketTools.InternetMail.ThreadModelAttribute

[Visual Basic]
```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False, Inherited:=True)>
Public Class InternetMail.ThreadModelAttribute
    Inherits Attribute
```

[C#]
```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False, Inherited=True)]
public class InternetMail.ThreadModelAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The ThreadModel attribute is used to define the threading model that is to be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#.

Example

```
<Assembly:
SocketTools.InternetMail.ThreadModel(SocketTools.InternetMail.ThreadModelAttribute.Model.SingleThread)>
```

```
[assembly:
SocketTools.InternetMail.ThreadModel(SocketTools.InternetMail.ThreadModelAttribute.Model.SingleThread)]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

InternetMail.ThreadModelAttribute Members | SocketTools Namespace | ThreadModel Property (SocketTools.InternetMail)

InternetMail.ThreadModelAttribute Members

InternetMail.ThreadModelAttribute overview

Public Instance Constructors

InternetMail.ThreadModelAttribute Constructor	Constructor for the ThreadModel attribute which defines the threading model.

Public Instance Properties

ThreadModel	Returns the threading model used by the class.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetMail.ThreadModelAttribute Class | SocketTools Namespace | ThreadModel Property (SocketTools.InternetMail)

InternetMail.ThreadModelAttribute Constructor

Constructor for the ThreadModel attribute which defines the threading model.

[Visual Basic]

```
Public Sub New( _
   ByVal threadModel As Model _
)
```

[C#]

```
public InternetMail.ThreadModelAttribute(
   Model threadModel
);
```

Parameters

threadModel

A Model enumeration value which specifies the threading model which will be used when creating an instance of the class. A value of zero specifies a single threaded model, while a non-zero value specifies a free threaded model.

Remarks

The **ThreadModel** attribute specifies the threading model that is used by the class instance when a connection is established. The default threading model is single threaded, which specifies that only the thread that established the connection should be permitted to invoke methods.

It is important to note that the single threading model does not limit the application to a single thread of execution. When a session is established using the **Connect** method, that session is attached to the thread that created it. From that point on, until the session is terminated, only the owner may invoke methods in that instance of the class. The ownership of the class instance may be transferred from one thread to another using the **AttachThread** method.

Setting this attribute to a non-zero value disables certain internal safety checks that are performed by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads. The application must ensure that no two threads will attempt to invoke a blocking method at the same time. In other words, if one thread invokes a method, the application must ensure that another thread will not attempt to invoke any other method at the same time using the same instance of the class.

Changing the value of this attribute will also change the default value for the **ThreadModel** property for all instances of the class.

See Also

InternetMail.ThreadModelAttribute Class | SocketTools Namespace

InternetMail.ThreadModelAttribute Properties

The properties of the **InternetMail.ThreadModelAttribute** class are listed below. For a complete list of **InternetMail.ThreadModelAttribute** class members, see the InternetMail.ThreadModelAttribute Members topic.

Public Instance Properties

ThreadModel	Returns the threading model used by the class.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

InternetMail.ThreadModelAttribute Class | SocketTools Namespace | ThreadModel Property (SocketTools.InternetMail)

InternetMail.ThreadModelAttribute.ThreadModel Property

Returns the threading model used by the class.

[Visual Basic]

```
Public Property ThreadModel As Model
```

[C#]

```
public InternetMail.ThreadModelAttribute.Model ThreadModel {get; set;}
```

Property Value

A Model enumeration value which specifies the threading model which will be used when an instance of the class is created.

See Also

InternetMail.ThreadModelAttribute Class | SocketTools Namespace

InternetMailException Class

The exception that is thrown when a client error occurs.

For a list of all members of this type, see InternetMailException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.InternetMailException

[Visual Basic]

```
Public Class InternetMailException
    Inherits ApplicationException
```

[C#]

```
public class InternetMailException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A InternetMailException is thrown by the InternetMail class when an error occurs.

The default constructor for the InternetMailException class sets the **ErrorCode** property to the last client error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetMail (in SocketTools.InternetMail.dll)

See Also

InternetMailException Members | SocketTools Namespace

InternetMailException Members

InternetMailException overview

Public Instance Constructors

InternetMailException	Overloaded. Initializes a new instance of the InternetMailException class.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetMailException Class | SocketTools Namespace

InternetMailException Constructor

Initializes a new instance of the InternetMailException class with the last network error code.

Overload List

Initializes a new instance of the InternetMailException class with the last network error code.

public InternetMailException();

Initializes a new instance of the InternetMailException class with a specified error number.

public InternetMailException(int);

Initializes a new instance of the InternetMailException class with a specified error message.

public InternetMailException(string);

Initializes a new instance of the InternetMailException class with a specified error message and a reference to the inner exception that is the cause of this exception.

public InternetMailException(string,Exception);

See Also

InternetMailException Class | SocketTools Namespace

InternetMailException Constructor ()

Initializes a new instance of the InternetMailException class with the last network error code.

[Visual Basic]

```
Overloads Public Sub New()
```

[C#]

```
public InternetMailException();
```

Remarks

The ctor constructor sets the **ErrorCode** property to the last client error that occurred. For more information about the errors that may occur, refer to the InternetMail.ErrorCode enumeration.

See Also

InternetMailException Class | SocketTools Namespace | InternetMailException Constructor Overload List

InternetMailException Constructor (String)

Initializes a new instance of the InternetMailException class with a specified error message.

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal message As String _
)
```

```
[C#]
public InternetMailException(
   string message
);
```

Parameters

message
The error message that explains the reason for the exception.

Remarks

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

InternetMailException Class | SocketTools Namespace | InternetMailException Constructor Overload List

InternetMailException Constructor (String, Exception)

Initializes a new instance of the InternetMailException class with a specified error message and a reference to the inner exception that is the cause of this exception.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal innerException As Exception _
)
```

[C#]

```
public InternetMailException(
   string message,
   Exception innerException
);
```

Parameters

message

The error message that explains the reason for the exception.

innerException

The exception that is the cause of the current exception. If the ***innerException*** parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

InternetMailException Class | SocketTools Namespace | InternetMailException Constructor Overload List

InternetMailException Constructor (Int32)

Initializes a new instance of the InternetMailException class with a specified error number.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

[C#]

```
public InternetMailException(
   int code
);
```

Parameters

code

An integer value which specifies an error code.

Remarks

This constructor sets the **ErrorCode** property to the specified error code. For more information about the errors that may occur, refer to the InternetMail.ErrorCode enumeration.

See Also

InternetMailException Class | SocketTools Namespace | InternetMailException Constructor Overload List

InternetMailException Properties

The properties of the **InternetMailException** class are listed below. For a complete list of **InternetMailException** class members, see the InternetMailException Members topic.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

InternetMailException Class | SocketTools Namespace

InternetMailException.ErrorCode Property

Gets a value which specifies the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property ErrorCode As ErrorCode
```

[C#]

```
public InternetMail.ErrorCode ErrorCode {get;}
```

Property Value

Returns a InternetMail.ErrorCode enumeration value which specifies the error code.

Remarks

The **ErrorCode** property returns the error code that specifies the cause of the exception.

The default constructor for the InternetMailException class sets the error code to the last network error that occurred. For more information about the errors that may occur, refer to the InternetMail.ErrorCode enumeration.

See Also

InternetMailException Class | SocketTools Namespace

InternetMailException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]

```
Overrides Public ReadOnly Property Message As String
```

[C#]

```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

InternetMailException Class | SocketTools Namespace

InternetMailException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception. This value is the same as the values returned by the Windows Sockets API. For more information about socket error codes, see the Windows Socket Version 2 API error code documentation in MSDN.

See Also

InternetMailException Class | SocketTools Namespace

InternetMailException Methods

The methods of the **InternetMailException** class are listed below. For a complete list of **InternetMailException** class members, see the InternetMailException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetMailException Class | SocketTools Namespace

InternetMailException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]
```
Overrides Public Function ToString() As String
```

[C#]
```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

InternetMailException Class | SocketTools Namespace

InternetServer Class

A general purpose class for developing Internet server applications.

For a list of all members of this type, see InternetServer Members.

System.Object
  SocketTools.InternetServer

[Visual Basic]
```
Public Class InternetServer
    Implements IDisposable
```

[C#]
```
public class InternetServer : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The **InternetServer** class provides a simplified interface for creating event-driven, multithreaded server applications using the TCP/IP protocol. The class interface is similar to the **SocketWrench** class, however it is designed specifically to make it easier to implement a server application without requiring the need to manage multiple socket classes. In addition, the **InternetServer** class supports secure communications using the Transport Layer Security (TLS) protocol.

Each instance of the class represents a server, and each active client connection is managed internally and referenced by an integer value which uniquely identifies the client session. All interaction with the server and the clients connected to it uses an event-driven model, with the server application written to respond to events such as **OnConnect**, **OnRead** and **OnWrite**.

Developers who have used the **SocketWrench** class will find the **InternetServer** class has a familiar interface, with a subset of properties and methods that are specific to creating a server application. Each of the network events have an extra parameter which specifies the socket handle which should be used when communicating with the client. This enables the application to communicate with multiple clients without having to create multiple socket classes.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

InternetServer Members | SocketTools Namespace

InternetServer Members

InternetServer overview

Public Instance Constructors

InternetServer Constructor	Initializes a new instance of the InternetServer class.

Public Instance Fields

AdapterAddress	Returns the IP address associated with the specified network adapter.
ClientHandle	Return the socket handle associated with a specific client session.

Public Instance Properties

AdapterCount	Get the number of available local and remote network adapters.
Backlog	Gets and sets the number of client connections that may be queued by the server.
CertificateName	Gets and sets a value that specifies the name of the server certificate.
CertificatePassword	Gets and sets the password associated with the server certificate.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateUser	Gets and sets the user that owns the server certificate.
ClientAddress	Gets a value that specifies the Internet address of the current client session.
ClientCount	Gets the number of active client sessions connected to the server.
ClientHost	Gets a value that specifies the hostname for the current client session.
ClientId	Gets the unique client identifier for the current client session.
ClientName	Gets and sets a unique string moniker that is associated with the current client session.
ClientPort	Gets a value that specifies the port number used by the current client session.
ClientThread	Gets the thread ID for the current client session.
CodePage	Gets and sets the code page used when reading and writing text.
ExternalAddress	Gets a value that specifies the external Internet

	address for the local system.
IdleTime	Gets a value which specifies the amount of time a socket has been idle
IsActive	Gets a value which indicates if the server is active.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking socket operation.
IsClosed	Gets a value which indicates if the connection to the client has been closed.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsListening	Gets a value which indicates if the server is listening for client connections.
IsLocked	Gets a value which indicates if the server has been locked.
IsReadable	Gets a value which indicates if there is data available to be read from the current client.
IsWritable	Gets a value which indicates if data can be written to the current client without blocking.
KeepAlive	Gets and sets a value which indicates if keep-alive packets are sent on a connected socket.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
MaxClients	Gets and sets the maximum number of clients that can connect to the server.
NoDelay	Gets and sets a value which specifies if the Nagle algorithm should be enabled or disabled.
Options	Gets and sets a value which specifies one or more server options.
Priority	Gets and sets a value which specifies the server priority.
ReuseAddress	Gets and sets a value which indicates if the server address can be reused.
Secure	Gets and sets a value which specifies if client connections are secure.
SecureProtocol	Gets and sets a value which specifies the protocol used for secure client connections.
ServerAddress	Gets and sets the address that will be used by the server to listen for connections.
ServerHandle	Gets the handle to the socket created to listen for client connections.

ServerName	Gets a value which specifies the host name for the local system.
ServerPort	Gets and sets the port number that will be used by the server to listen for connections.
ServerThread	Gets the thread ID for the current server.
StackSize	Gets and sets the size of the stack allocated for threads created by the server.
Status	Gets a value which specifies the current status of the server.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the network function tracing logfile.
TraceFlags	Gets and sets a value which specifies the network function tracing flags.
Version	Gets a value which returns the current version of the InternetServer class library.

Public Instance Methods

Abort	Overloaded. Abort the connection with a remote host.
Broadcast	Overloaded. Broadcast data to all active clients connected to the server
Cancel	Overloaded. Cancel the current blocking socket operation.
Disconnect	Overloaded. Disconnect the specified client connection from the server.
Dispose	Overloaded. Releases all resources used by InternetServer.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
FindClient	Overloaded. Return the socket handle for the client session with the specified moniker.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the

	InternetServer class.
Lock	Lock the server to synchronize access to shared data for all active client sessions.
Peek	Overloaded. Read data from the client and store it in a byte array, but do not remove the data from the socket buffers.
Read	Overloaded. Read data from the client socket and store it in a byte array.
ReadLine	Overloaded. Read up to a line of data from the client and return it in a string buffer.
Reject	Overloaded. Rejects a connection request from a client.
Reset	Reset the internal state of the object, resetting all properties to their default values.
Resolve	Resolves a host name to a host IP address.
Restart	Restarts the server and terminates all active client connections.
Resume	Resume accepting new client connections.
Start	Overloaded. Start listening for client connections on the specified IP address and port number.
Stop	Stop listening for new client connections and terminate all active clients already connected to the server.
Suspend	Overloaded. Suspend accepting new client connections with additional options.
Throttle	Overloaded. Limit the maximum number of client connections, connections per IP address and connection rate.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the server.
Unlock	Unlock the server and allow other server threads to resume execution.
Write	Overloaded. Write one or more bytes of data to a client.
WriteLine	Overloaded. Send a line of text to a client, terminated by a carriage-return and linefeed.

Public Instance Events

OnAccept	Occurs when a client attempts to establish a connection with the server.
OnCancel	Occurs when a blocking socket operation is canceled.

OnConnect	Occurs when a connection is established with the remote host.
OnDisconnect	Occurs when the remote host disconnects from the local system.
OnError	Occurs when an socket operation fails.
OnIdle	Occurs when the there are no clients connected to the server.
OnRead	Occurs when data is available to be read from the client.
OnStart	Occurs when the server starts accepting connections.
OnStop	Occurs when the server stops accepting connections.
OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
OnWrite	Occurs when data can be written to the client.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the InternetServer class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetServer Class | SocketTools Namespace

InternetServer Constructor

Initializes a new instance of the InternetServer class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public InternetServer();
```

Example

The following example demonstrates creating an instance of the **InternetServer** class object and starting a server using the Start method.

```
Dim Server As SocketTools.InternetServer
Dim strLocalAddress As String
Dim nLocalPort As Integer

Server = New SocketTools.InternetServer

strLocalAddress = TextBox1.Text.Trim()
nLocalPort = Val(TextBox2.Text)

If Server.Start(strLocalAddress, nLocalPort) Then
    StatusBar1.Text = "The server has started listening for connections"
Else
    StatusBar1.Text = "The server could not be started"
End If
```

See Also

InternetServer Class | SocketTools Namespace

InternetServer Fields

The fields of the **InternetServer** class are listed below. For a complete list of **InternetServer** class members, see the InternetServer Members topic.

Public Instance Fields

◆ AdapterAddress	Returns the IP address associated with the specified network adapter.
◆ ClientHandle	Return the socket handle associated with a specific client session.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.AdapterAddress Field

Returns the IP address associated with the specified network adapter.

[Visual Basic]

```
Public ReadOnly AdapterAddress As AdapterAddressArray
```

[C#]

```
public readonly AdapterAddressArray AdapterAddress;
```

Remarks

The **AdapterAddress** array returns the IP addresses that are associated with the local network or remote dial-up network adapters configured on the system. The **AdapterCount** property can be used to determine the number of adapters that are available.

Multihomed systems with more than one local network adapter, or a combination of local and dial-up adapters will not be listed in a specific order. An application should not make the assumption that the first address returned by **AdapterAddress** always refers to a local network adapter.

Note that it is possible that the **AdapterCount** property will return 0, and **AdapterAddress** will return an empty string. This indicates that the system does not have a physical network adapter with an assigned IP address, and there are no dial-up networking connections currently active. If a dial-up networking connection is established at some later point, the **AdapterCount** property will change to 1, and the **AdapterAddress** property will return the IP address allocated for that connection.

See Also

InternetServer Class | SocketTools Namespace | AdapterAddressArray Class | AdapterCount Property

InternetServer.ClientHandle Field

Return the socket handle associated with a specific client session.

[Visual Basic]
```
Public ReadOnly ClientHandle As ClientHandleArray
```

[C#]
```
public readonly ClientHandleArray ClientHandle;
```

Remarks

The **ClientHandle** array is a read-only, zero-based property array that returns the socket handle allocated for the client session specified by the Index parameter. An exception will be thrown if the index value exceeds the maximum number of active client sessions. To determine the number of clients that are currently connected to the server, use the **ClientCount** property.

You should always check the value of the **ClientCount** property prior to enumerating through the client connections using the **ClientHandle** array. Never assume that a particular client session will always be found in the same position in the array. The socket handles returned by the array can be used in conjunction with the **Read** and **Write** methods to exchange data with a particular client session outside of an event handler.

The index into this array may also be a string which specifies the name of a client session, as set by the **ClientName** property. This can be a convenient way to obtain the socket handle for a specific client by name, rather than a numeric index.

See Also

InternetServer Class | SocketTools Namespace

InternetServer Properties

The properties of the **InternetServer** class are listed below. For a complete list of **InternetServer** class members, see the InternetServer Members topic.

Public Instance Properties

AdapterCount	Get the number of available local and remote network adapters.
Backlog	Gets and sets the number of client connections that may be queued by the server.
CertificateName	Gets and sets a value that specifies the name of the server certificate.
CertificatePassword	Gets and sets the password associated with the server certificate.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateUser	Gets and sets the user that owns the server certificate.
ClientAddress	Gets a value that specifies the Internet address of the current client session.
ClientCount	Gets the number of active client sessions connected to the server.
ClientHost	Gets a value that specifies the hostname for the current client session.
ClientId	Gets the unique client identifier for the current client session.
ClientName	Gets and sets a unique string moniker that is associated with the current client session.
ClientPort	Gets a value that specifies the port number used by the current client session.
ClientThread	Gets the thread ID for the current client session.
CodePage	Gets and sets the code page used when reading and writing text.
ExternalAddress	Gets a value that specifies the external Internet address for the local system.
IdleTime	Gets a value which specifies the amount of time a socket has been idle
IsActive	Gets a value which indicates if the server is active.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking socket operation.
IsClosed	Gets a value which indicates if the connection to the client has been closed.

IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsListening	Gets a value which indicates if the server is listening for client connections.
IsLocked	Gets a value which indicates if the server has been locked.
IsReadable	Gets a value which indicates if there is data available to be read from the current client.
IsWritable	Gets a value which indicates if data can be written to the current client without blocking.
KeepAlive	Gets and sets a value which indicates if keep-alive packets are sent on a connected socket.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
MaxClients	Gets and sets the maximum number of clients that can connect to the server.
NoDelay	Gets and sets a value which specifies if the Nagle algorithm should be enabled or disabled.
Options	Gets and sets a value which specifies one or more server options.
Priority	Gets and sets a value which specifies the server priority.
ReuseAddress	Gets and sets a value which indicates if the server address can be reused.
Secure	Gets and sets a value which specifies if client connections are secure.
SecureProtocol	Gets and sets a value which specifies the protocol used for secure client connections.
ServerAddress	Gets and sets the address that will be used by the server to listen for connections.
ServerHandle	Gets the handle to the socket created to listen for client connections.
ServerName	Gets a value which specifies the host name for the local system.
ServerPort	Gets and sets the port number that will be used by the server to listen for connections.
ServerThread	Gets the thread ID for the current server.
StackSize	Gets and sets the size of the stack allocated for threads created by the server.
Status	Gets a value which specifies the current status of

	the server.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the network function tracing logfile.
TraceFlags	Gets and sets a value which specifies the network function tracing flags.
Version	Gets a value which returns the current version of the InternetServer class library.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.AdapterCount Property

Get the number of available local and remote network adapters.

[Visual Basic]
```
Public ReadOnly Property AdapterCount As Integer
```

[C#]
```
public int AdapterCount {get;}
```

Property Value

Returns the number of available local and remote network adapters.

Remarks

The **AdapterCount** property returns the number of local and remote dial-up networking adapters available on the local system. This value can be used in conjunction with the **AdapterAddress** array to enumerate the IP addresses assigned to the various network adapters.

Note that it is possible that the **AdapterCount** property will return 0, and **AdapterAddress** will return an empty string. This indicates that the system does not have a physical network adapter with an assigned IP address, and there are no dial-up networking connections currently active. If a dial-up networking connection is established at some later point, the **AdapterCount** property will change to 1, and the **AdapterAddress** property will return the IP address allocated for that connection.

See Also

InternetServer Class | SocketTools Namespace | AdapterAddress Field

InternetServer.Backlog Property

Gets and sets the number of client connections that may be queued by the server.

[Visual Basic]
```
Public Property Backlog As Integer
```

[C#]
```
public int Backlog {get; set;}
```

Property Value

Returns an integer value that specifies the size of the backlog queue. The default value is 5.

Remarks

The **Backlog** property specifies the maximum size of the queue used to manage pending connections to the service. If the property is set to value which exceeds the maximum size for the underlying service provider, it will be silently adjusted to the nearest legal value. There is no standard way to determine what the maximum backlog value is.

This property should be set to the desired value before the **Start** method is called. The default backlog value is 5 on all Windows platforms. The Windows Server platforms support a maximum backlog value of 200.

Note that this property does not specify the total number of connections that the server application may accept. It only specifies the size of the backlog queue which is used to manage pending client connections. Once the client connection has been accepted, it is removed from the queue. Set the **MaxClients** property to specify the maximum number of clients that may connect with the server.

See Also

InternetServer Class | SocketTools Namespace | IsListening Property | MaxClients Property | Start Method

InternetServer.CertificateName Property

Gets and sets a value that specifies the name of the server certificate.

[Visual Basic]

```
Public Property CertificateName As String
```

[C#]

```
public string CertificateName {get; set;}
```

Property Value

A string which specifies the server certificate name.

Remarks

The **CertificateName** property sets the common name or friendly name of the certificate that should be used when starting a secure server. If the **Secure** property is set to True, this property must be specify a valid certificate name. The certificate must have a private key associated with it, otherwise client connections will fail because the class will be unable to create a security context for the session.

When the certificate store is searched for a matching certificate, it will first search for any certificate with a friendly name that matches the property value. If no valid certificate is found, it will then search for a certificate with a matching common name.

Certificates may be installed and viewed on the local system using the Certificate Manager that is included with the Windows operating system. For more information, refer to the documentation for the Microsoft Management Console.

See Also

InternetServer Class | SocketTools Namespace | CertificateStore Property | Secure Property

InternetServer.CertificateStore Property

Gets and sets a value that specifies the name of the local certificate store.

[Visual Basic]
```
Public Property CertificateStore As String
```

[C#]
```
public string CertificateStore {get; set;}
```

Property Value

A string which specifies the certificate store name. The default value is the current user's personal certificate store.

Remarks

The **CertificateStore** property is used to specify the name of the certificate store which contains the security certificate to use when security is enabled for the server. The certificate may either be stored in the registry or in a file. If the certificate is stored in the registry, then this property should be set to one of the following predefined values:

Store Name	Description
CA	Certification authority certificates. These are certificates that are issued by entities which are entrusted to issue certificates to other individuals or organizations. Companies such as VeriSign and Thawte act as certification authorities.
MY	Personal certificates and their associated private keys for the current user. This store typically holds the client certificates used to establish a user's credentials. If a certificate store is not specified, this is the default value that is used.
ROOT	Certificates that have been self-signed by a certificate authority. Root certificates for a number of different certification authorities such as VeriSign and Thawte are installed as part of the operating system and periodically updated by Microsoft.

In most cases the server certificate will be installed in the user's personal certificate store, and therefore it is not necessary to set this property value because that is the default location that will be used to search for the certificate. This property is only used if the **CertificateName** property is also set to a valid certificate name.

If you are using a local certificate store, with the certificate and private key stored in the registry, you can explicitly specify whether the certificate store for the current user or the local machine (all users) should be used. This is done by prefixing the certificate store name with "HKCU" for the current user, or "HKLM" for the local machine. For example, a certificate store name of "HKLM:MY" would specify the personal certificate store for the local machine, rather than the current user. If neither prefix is specified, it will default to the certificate store for the current user.

This property may also be used to specify a file that contains the server certificate. In this case, the property should specify the full path to the file and must contain both the certificate and private key in

PKCS12 format. If the file is protected by a password, the **CertificatePassword** property must also be set to specify the password.

See Also

InternetServer Class | SocketTools Namespace | CertificatePassword Property | Secure Property

InternetServer.ClientAddress Property

Gets a value that specifies the Internet address of the current client session.

[Visual Basic]

```
Public ReadOnly Property ClientAddress As String
```

[C#]

```
public string ClientAddress {get;}
```

Property Value

A string which specifies an Internet address in dotted notation.

Remarks

The **ClientAddress** property returns the address of the current client session which has connected to the server. This property only returns a meaningful value inside an event handler such as **OnAccept** or **OnConnect**.

If this property is accessed inside an **OnAccept** event handler, it will return the address of the client that is requesting the connection. The server application may use this information to determine if it wishes to accept or reject the client connection.

See Also

InternetServer Class | SocketTools Namespace | ClientHost Property | ClientPort Property

InternetServer.ClientCount Property

Gets the number of active client sessions connected to the server.

[Visual Basic]

```
Public ReadOnly Property ClientCount As Integer
```

[C#]

```
public int ClientCount {get;}
```

Property Value

An integer value which specifies the number of active client sessions.

Remarks

The **ClientCount** read-only property returns the number of active client sessions that have been established with the server. This property is typically used in conjunction with the **ClientHandle** array to enumerate the handles for all client sessions.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.ClientHost Property

Gets a value that specifies the hostname for the current client session.

[Visual Basic]

```
Public ReadOnly Property ClientHost As String
```

[C#]

```
public string ClientHost {get;}
```

Property Value

A string which specifies the peer host name.

Remarks

The **ClientHost** property returns the hostname of the current client session which has established a connection with the server. This property value is only meaningful when accessed within an event handler, such as the **OnConnect** event.

Accessing this property causes the class to perform a blocking reverse DNS lookup, attempting to match the client Internet address with a hostname. Not all addresses have a reverse DNS record, in which case this property will return an empty string. It is recommended that most applications use the value of the **ClientAddress** property rather than use the **ClientHost** property to distinguish between client connections.

See Also

InternetServer Class | SocketTools Namespace | ClientAddress Property | ClientPort Property

InternetServer.ClientId Property

Gets the unique client identifier for the current client session.

[Visual Basic]

```
Public ReadOnly Property ClientId As Integer
```

[C#]

```
public int ClientId {get;}
```

Property Value

An integer value which uniquely identifies the client session.

Remarks

Each client connection that is accepted by the server is assigned a unique numeric value. This value can be used by the application to identify that client session, and is different than the socket handle allocated for the client. While it is possible for a client socket handle to be reused by the operating system, client IDs are unique throughout the life of the server session and are never duplicated.

It is important to note that the actual value of the client ID should be considered opaque. It is only guaranteed that the value will be greater than zero, and that it will be unique to the client session.

This property only returns a meaningful value when accessed from within a class event handler, or a method that has been invoked from within an event handler.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.ClientName Property

Gets and sets a unique string moniker that is associated with the current client session.

[Visual Basic]

```
Public Property ClientName As String
```

[C#]

```
public string ClientName {get; set;}
```

Property Value

A string moniker which uniquely identifies the client session. If no moniker has been specified for the client session, this property will return an empty string.

Remarks

A client moniker is a string which can be used to uniquely identify a specific client session aside from its socket handle. A moniker can be assigned to the client session by setting the **ClientName** property from within a class event handler such as the **OnConnect** event.

Monikers are not case-sensitive, and they must be unique so that no client socket for a particular server can have the same moniker. The maximum length for a moniker is 127 characters.

This property only returns a meaningful value when accessed from within a class event handler, or a method that has been invoked from within an event handler.

Example

The following example sets the moniker for the client session in the OnConnect event handler.

```
private void Server1_OnConnect(object sender,
SocketTools.InternetServer.ConnectEventArgs e)
{
    Server1.ClientName = "Client" + Server1.ClientId.ToString();
}
```

See Also

InternetServer Class | SocketTools Namespace

InternetServer.ClientPort Property

Gets a value that specifies the port number used by the current client session.

[Visual Basic]

```
Public ReadOnly Property ClientPort As Integer
```

[C#]

```
public int ClientPort {get;}
```

Property Value

An integer value which specifies the peer port number.

Remarks

The **ClientPort** property returns the port number that the current client has used when establishing a connection with the server. This property value is only meaningful when accessed within an event handler such as the **OnConnect** event.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.ClientThread Property

Gets the thread ID for the current client session.

[Visual Basic]

```
Public ReadOnly Property ClientThread As Integer
```

[C#]

```
public int ClientThread {get;}
```

Property Value

An integer value which identifies the client thread that was created to manage the client session.

Remarks

Until the thread terminates, the thread identifier uniquely identifies the thread throughout the system.

This property only returns a meaningful value when accessed from within a class event handler, or a method that has been invoked from within an event handler.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.CodePage Property

Gets and sets the code page used when reading and writing text.

[Visual Basic]
```
Public Property CodePage As Integer
```

[C#]
```
public int CodePage {get; set;}
```

Property Value

An integer value which specifies the current code page. A value of zero specifies the default code page for the current locale should be used. To preserve the original Unicode text, you can use code page 65001 which specifies UTF-8 character encoding.

Remarks

All data which is exchanged over a socket is sent and received as 8-bit bytes, typically referred to as "octets" in networking terminology. However, strings in .NET are Unicode where each character is represented by 16 bits. To send and receive data using strings, these Unicode strings are converted to a stream of bytes.

By default, strings are converted to an array of bytes using the code page for the current locale, mapping the 16-bit Unicode characters to bytes. Similarly, when reading data from the socket into a string buffer, the stream of bytes received from the remote host are converted to Unicode before they are returned to your application.

If you are exchanging text with another system and it appears to corrupted or characters are being replaced with question marks or other symbols, it is likely the system is sending text which is using a different character encoding. Most services use UTF-8 encoding to represent non-ASCII characters and selecting the UTF-8 code page will typically resolve the issue.

Strings are only guaranteed to be safe when sending and receiving text. Using a string data type is not recommended when reading or writing binary data to a socket. If possible, you should always use a byte array as the buffer parameter for the Read and Write methods whenever you are exchanging binary data.

For backwards compatibility, this class defaults to using the code page for the current locale. This property value directly corresponds to Windows code page identifiers, and will accept any valid code page supported by the .NET Framework. Setting this property to an invalid code page will generate an exception.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.ExternalAddress Property

Gets a value that specifies the external Internet address for the local system.

[Visual Basic]

```
Public ReadOnly Property ExternalAddress As String
```

[C#]

```
public string ExternalAddress {get;}
```

Property Value

A string which specifies an Internet address using dotted notation.

Remarks

The **ExternalAddress** property returns the IP address assigned to the router that connects the local host to the Internet. This is typically used by an application executing on a system in a local network that uses a router which performs Network Address Translation (NAT). The **ExternalAddress** property can be used to determine the IP address assigned to the router on the Internet side of the connection and can be particularly useful for servers running on a system behind a NAT router.

Using this property requires that you have an active connection to the Internet; checking the value of this property on a system that uses dial-up networking may cause the operating system to automatically connect to the Internet service provider. The class may be unable to determine the external IP address for the local host for a number of reasons, particularly if the system is behind a firewall or uses a proxy server that restricts access to external sites on the Internet. If the external address for the local host cannot be determined, the property will return an empty string.

If the class is able to obtain a valid external address for the local host, that address will be cached for sixty minutes. Because dial-up connections typically have different IP addresses assigned to them each time the system is connected to the Internet, it is recommended that this property only be used in conjunction with broadband connections using a NAT router.

It is important to note that checking this property value may cause the current thread to block until the external IP address can be resolved.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.IsBlocked Property

Gets a value which indicates if the current thread is performing a blocking socket operation.

[Visual Basic]
```
Public ReadOnly Property IsBlocked As Boolean
```

[C#]
```
public bool IsBlocked {get;}
```

Property Value

Returns **true** if the current thread is blocking, otherwise returns **false**.

Remarks

The **IsBlocked** property returns **true** if the current thread is blocked performing an operation. Because the Windows Sockets API only permits one blocking operation per thread of execution, this property should be checked before starting any blocking operation in response to an event.

If the **IsBlocked** property returns **false**, this means there are no blocking operations on the current thread at that time. However, this does not guarantee that the next socket operation will not fail. An application should always check the return value from a socket operation and check the value of the **LastError** property if an error occurs.

Note that this property will return **true** if there is any blocking operation being performed by the current thread, regardless of whether this specific instance of the class is responsible for the blocking operation or not.

This property only returns a meaningful value when accessed from within a class event handler, or a method that has been invoked from within an event handler.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.IsClosed Property

Gets a value which indicates if the connection to the client has been closed.

[Visual Basic]

```
Public ReadOnly Property IsClosed As Boolean
```

[C#]

```
public bool IsClosed {get;}
```

Property Value

Returns **true** if the connection has been closed; otherwise returns **false**.

Remarks

This property only returns a meaningful value when accessed from within a class event handler, or a method that has been invoked from within an event handler.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.IsInitialized Property

Gets a value which indicates if the current instance of the class has been initialized successfully.

[Visual Basic]

```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]

```
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the class instance has been initialized; otherwise returns **false**.

Remarks

The **IsInitialized** property is used to determine if the current instance of the class has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the class may not be able to initialize itself.

The most common reasons that a class instance may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required networking libraries or not being able to access the system registry.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.IsListening Property

Gets a value which indicates if the server is listening for client connections.

[Visual Basic]

```
Public ReadOnly Property IsListening As Boolean
```

[C#]

```
public bool IsListening {get;}
```

Property Value

Returns **true** if the server is listening for client connections; otherwise returns **false**.

Remarks

The **IsListening** property will return **true** if the **Start** method was called and the server is currently accepting incoming client connections. In all other situations, this property will return **false**.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.IsLocked Property

Gets a value which indicates if the server has been locked.

[Visual Basic]
```
Public ReadOnly Property IsLocked As Boolean
```

[C#]
```
public bool IsLocked {get;}
```

Property Value

Returns **true** if the server has been locked; otherwise returns **false**.

Remarks

The **IsLocked** property returns **true** if the server has been locked using the **Lock** method. When a server is locked, all background threads created by the server will block, waiting for the lock to be released. If this property returns a value of **true**, no client connections can be accepted by the server, and no network events will be generated.

The **Lock** method creates a critical section which prevents other threads from performing any network operation. This is useful when the program needs to update global data and wants to ensure that no network operations occur while the data is being modified. However, applications must take care to release the lock as quickly as possible. If a function locks the server, it must make sure that it releases the lock before exiting that function. Leaving the server locked across function calls or event handlers can result in the server becoming non-responsive.

See Also

InternetServer Class | SocketTools Namespace | Lock Method | Unlock Method

InternetServer.IsReadable Property

Gets a value which indicates if there is data available to be read from the current client.

[Visual Basic]
```
Public ReadOnly Property IsReadable As Boolean
```

[C#]
```
public bool IsReadable {get;}
```

Property Value

Returns **true** if there is data available to be read; otherwise returns **false**.

Remarks

The **IsReadable** property returns **true** if data can be read from the current client session without causing the current thread to block. Note that even if this property does return **true** indicating that there is data available to be read, applications should always check the return value from the **Read** method.

This property only returns a meaningful value when accessed from within a class event handler, or a method that has been invoked from within an event handler.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.IsWritable Property

Gets a value which indicates if data can be written to the current client without blocking.

[Visual Basic]

```
Public ReadOnly Property IsWritable As Boolean
```

[C#]

```
public bool IsWritable {get;}
```

Property Value

Returns **true** if data can be written to the client; otherwise returns **false**.

Remarks

The **IsWritable** property returns **true** if data can be written to the client without causing the current thread to block. Note that even if this property does return **true** indicating that data can be written to the client, applications should always check the return value from the **Write** method.

This property only returns a meaningful value when accessed from within a class event handler, or a method that has been invoked from within an event handler.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.KeepAlive Property

Gets and sets a value which indicates if keep-alive packets are sent on a connected socket.

[Visual Basic]

```
Public Property KeepAlive As Boolean
```

[C#]

```
public bool KeepAlive {get; set;}
```

Property Value

Returns **true** if keep-alive packets are enabled, otherwise returns **false**. The default value is **false**.

Remarks

Setting the **KeepAlive** property to a value of **true** specifies that special packets are to be sent to the remote system when no data is being exchanged to ensure the connection remains active. This property can only be set for sockets that were created with the **Protocol** property set to a value of **SocketProtocol.protocolStream**.

It is important to note that the system will not start generating keep-alive packets until two hours after it has been enabled, so this option is only relevant for connections that will be maintained for long periods of time. The actual interval for the keep-alive period can only be changed in the system registry and affects all sockets, system-wide. For more information, refer to Microsoft Knowledge Base article 314053.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public InternetServer.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.MaxClients Property

Gets and sets the maximum number of clients that can connect to the server.

[Visual Basic]

```
Public Property MaxClients As Integer
```

[C#]

```
public int MaxClients {get; set;}
```

Property Value

An integer value which specifies the maximum number of client sessions that will be accepted by the server. A value of zero specifies that there is no fixed limit to the maximum number of clients.

Remarks

The **MaxClients** property specifies the maximum number of client connections that will be accepted by the server. Once the maximum number of connections has been established, the server will reject any subsequent connections until the number of active client connections drops below the specified value. A value of zero specifies that there should be no limit on the number of clients.

Changing the value of this property while a server is actively listening for connections will modify the maximum number of client connections permitted, but it will not affect connections that have already been established.

By default, there are no limits on the number of client connections or the connection rate when a server is started. Use the **Throttle** method to change the maximum number of client connections per IP address or the overall connection rate threshold for the server.

It is important to note that regardless of the maximum number of clients specified by this property, the actual number of client connections that can be managed by the server depends on the number of sockets that can be allocated from the operating system. The amount of physical memory installed on the system affects the number of connections that can be maintained because each connection allocates memory for the socket context from the non-paged memory pool.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.NoDelay Property

Gets and sets a value which specifies if the Nagle algorithm should be enabled or disabled.

[Visual Basic]
```
Public Property NoDelay As Boolean
```

[C#]
```
public bool NoDelay {get; set;}
```

Property Value

Returns **true** if the Nagle algorithm has been disabled; otherwise it returns **false**. The default value is **false**.

Remarks

The **NoDelay** property is used to enable or disable the Nagle algorithm, which buffers unacknowledged data and insures that a full-size packet can be sent to the remote host. By default this property value is set to **false**, which enables the Nagle algorithm (in other words, the data being written may not actually be sent until it is optimal to do so). Setting this property to **true** disables the Nagle algorithm, maintaining the time delays between the data packets being sent.

This property should be set to **true** only if it is absolutely required and the implications of doing so are understood. Disabling the Nagle algorithm can have a significant negative impact on the performance of your server.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.Options Property

Gets and sets a value which specifies one or more server options.

[Visual Basic]

```
Public Property Options As ServerOptions
```

[C#]

```
public InternetServer.ServerOptions Options {get; set;}
```

Property Value

Returns one or more ServerOptions enumeration flags which specify the options for the server. The default value for this property is **serverOptionNone**.

Remarks

The **Options** property specifies one or more default options which are used when starting the server using the **Start** method.

See Also

InternetServer Class | SocketTools Namespace | Start Method

InternetServer.Priority Property

Gets and sets a value which specifies the server priority.

[Visual Basic]

```
Public Property Priority As ServerPriority
```

[C#]

```
public InternetServer.ServerPriority Priority {get; set;}
```

Property Value

Returns a ServerPriority enumeration value which specifies the current server priority. The default value for this property is **priorityNormal**.

Remarks

The **Priority** property can be used to control the processor usage, memory and network bandwidth allocated by the server for client sessions. The default priority balances resource utilization and client throughput while ensuring that the user interface remains responsive to the user. Lower priorities reduce the overall resource utilization at the expense of throughput.

Higher priority values increases the thread priority and processor utilization for the client sessions. It is not recommended that you increase the server priority unless you understand the implications of doing so and have thoroughly tested your application. Raising the priority of the server can have a negative impact on the responsiveness of the user interface.

See Also

InternetServer Class | SocketTools Namespace | ServerPriority Enumeration

InternetServer.ReuseAddress Property

Gets and sets a value which indicates if the server address can be reused.

[Visual Basic]

```
Public Property ReuseAddress As Boolean
```

[C#]

```
public bool ReuseAddress {get; set;}
```

Property Value

Returns **true** if an address can be reused; otherwise returns **false**. The default value is **true**.

Remarks

The **ReuseAddress** property is used to determine if the local address and port number can be reused when starting a new instance of the server. Setting this property to **true** enables a server application to listen for connections using the specified address and port number even if they were in use recently. This is typically used to enable the server to close the listening socket and immediately reopen it without getting an error that the address is in use.

When a listening socket closed, the socket will normally go into a TIME-WAIT state where the local address and port number cannot be immediately reused. A consequence of this is that calling the **Stop** method immediately followed by the **Start** method using the same address and port number values may result in an error indicating that the specified address is already in use. By setting this property to **true**, that error is avoided and the listening socket can be created immediately without waiting for the TIME-WAIT period to elapse. Note that calling the **Restart** method allows the local address and port number to be reused, regardless of this property value.

If you wish to determine if a local port number is already in use by another application, set this property to **false** and attempt to start the server using that port number. If another application is already using that port number, an error will be generated indicating that the address is in use and the server could not be started.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.Secure Property

Gets and sets a value which specifies if client connections are secure.

[Visual Basic]
```
Public Property Secure As Boolean
```

[C#]
```
public bool Secure {get; set;}
```

Property Value

Returns **true** if a secure connections are enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Secure** property determines if client connections are encrypted using the standard SSL or TLS security protocols. The default value for this property is **false**, which specifies that clients will use a standard, unencrypted connection to the server. To enable secure connections, the application should set this property value to **true** prior to calling the **Start** method.

When secure connections are enabled, the server will accept the client connection and then wait for the client to initiate the handshake where both the client and server negotiate the various encryption options available. This process is handled automatically by the server, and all that is required is that the application specify the server certificate which should be used. This is done by setting the **CertificateName** property, and optionally the **CertificateStore** property if required.

It is recommended that the application use exception handling to catch any errors that may occur when changing the value of this property. If the class is unable to initialize the Windows security libraries, an exception will be thrown when this property value is modified.

See Also

InternetServer Class | SocketTools Namespace | CertificateName Property | CertificateStore Property | SecureProtocol Property

InternetServer.SecureProtocol Property

Gets and sets a value which specifies the protocol used for secure client connections.

[Visual Basic]
```
Public Property SecureProtocol As SecurityProtocols
```

[C#]
```
public InternetServer.SecurityProtocols SecureProtocol {get; set;}
```

Property Value

A SecurityProtocols enumeration value which identifies the protocol to be used when accepting a secure client connection.

Remarks

The **SecureProtocol** property can be used to specify the security protocol to be used when accepting a secure connection. By default, the class will attempt to use either SSL v3 or TLS v1 to accept the connection, with the appropriate protocol automatically selected based on the capabilities of the client. It is recommended that you only change this property value if you fully understand the implications of doing so. Assigning a value to this property will override the default protocol and force the class to attempt to use only the protocol specified.

Attempting to set this property after the server has been started will result in an exception being thrown. This property should only be set after setting the **Secure** property to **true** and before calling the **Start** method.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.ServerAddress Property

Gets and sets the address that will be used by the server to listen for connections.

[Visual Basic]

```
Public Property ServerAddress As String
```

[C#]

```
public string ServerAddress {get; set;}
```

Property Value

A string which specifies the IP address that the server will use to listen for incoming client connections. An empty string indicates that the server will accept connections on any valid network interface configured for the local system.

Remarks

The **ServerAddress** property is used to specify the default address that the server will use when listening for connections. Setting this property to the value 0.0.0.0 or an empty string indicates that the server should listen for client connections using any valid network interface. If an address is specified, it must be a valid Internet address that is bound to a network adapter configured on the local system. Clients will only be able to connect to the server using that specific address.

It is common to set this property to the value 127.0.0.1 for testing purposes. It is a non-routable address that specifies the local system, and most software firewalls are configured so they do not block applications using this address.

See Also

InternetServer Class | SocketTools Namespace | ServerName Property | ServerPort Property

InternetServer.ServerName Property

Gets a value which specifies the host name for the local system.

[Visual Basic]

```
Public ReadOnly Property ServerName As String
```

[C#]

```
public string ServerName {get;}
```

Property Value

A string which specifies the local host name.

Remarks

The **ServerName** property returns the fully-qualified host name assigned to the local system. This consists of the local computer name and its domain name. The actual value returned depends on the system configuration. If no domain has been specified for the system, then only the machine name will be returned.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.ServerPort Property

Gets and sets the port number that will be used by the server to listen for connections.

[Visual Basic]

```
Public Property ServerPort As Integer
```

[C#]

```
public int ServerPort {get; set;}
```

Property Value

An integer value which specifies the port number.

Remarks

The **ServerPort** property is used to set the port number that server will use to listen for incoming client connections. Valid port numbers are in the range of 1 to 65535. It is recommended that most custom servers specify a port number larger than 5000 to avoid potential conflicts with standard Internet services and ephemeral ports used by client applications.

If a port number is specified that is already in use by another application, the **OnError** event will fire and the background server thread will terminate. To enable a server to be stopped and immediately restarted using the same address and port number, make sure that the **ReuseAddress** property is set to a value of **true**.

See Also

InternetServer Class | SocketTools Namespace | ServerAddress Property | ServerName Property

InternetServer.ServerThread Property

Gets the thread ID for the current server.

[Visual Basic]
```
Public ReadOnly Property ServerThread As Integer
```

[C#]
```
public int ServerThread {get;}
```

Property Value

An integer value which identifies the server thread that was created. A return value of zero specifies that no server has been started.

Remarks

Until the thread terminates, the thread identifier uniquely identifies the thread throughout the system.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.Status Property

Gets a value which specifies the current status of the server.

[Visual Basic]

```
Public ReadOnly Property Status As ServerStatus
```

[C#]

```
public InternetServer.ServerStatus Status {get;}
```

Property Value

A ServerStatus enumeration value which specifies the current server status.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]

```
Public Property ThrowError As Boolean
```

[C#]

```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.Timeout Property

Gets and sets a value which specifies a timeout period in seconds.

[Visual Basic]

```
Public Property Timeout As Integer
```

[C#]

```
public int Timeout {get; set;}
```

Property Value

An integer value which specifies a timeout period in seconds.

Remarks

Setting the **Timeout** property specifies the number of seconds until a socket operation fails and returns an error.

For most server applications it is recommended the timeout period be set between 10 and 20 seconds. It is not recommended that you set the timeout period to zero.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.Trace Property

Gets and sets a value which indicates if network function logging is enabled.

[Visual Basic]
```
Public Property Trace As Boolean
```

[C#]
```
public bool Trace {get; set;}
```

Property Value

Returns **true** if network function tracing is enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Trace** property is used to enable (or disable) the tracing of network function calls. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the SocketTools networking classes and is enabled or disabled for an entire process. This means that once trace logging is enabled for a given component, all of the function calls made by the process using any of the SocketTools classes will be logged. For example, if you have an application using both the File Transfer Protocol and Post Office Protocol classes, and you set the **Trace** property to **true**, function calls made by both classes will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all classes used by the process.

If trace logging is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the logfile is fairly large. Since logfiles can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

When redistributing your application, make sure that you include the **SocketTools10.TraceLog.dll** module with your installation. If this library is not present, then no trace output will be generated and the value of the **Trace** property will be ignored. Only those function calls made by the SocketTools networking classes will be logged. Calls made directly to the Windows Sockets API, or calls made by other classes, will not be logged.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.TraceFile Property

Gets and sets a value which specifies the name of the network function tracing logfile.

[Visual Basic]
```
Public Property TraceFile As String
```

[C#]
```
public string TraceFile {get; set;}
```

Property Value

A string which specifies the name of the file.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named **SocketTools.log** is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFile** property should be set to the same value for each instance. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
 MyApp INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0 MyApp WRN:
connect(46, 192.0.0.1:1234, 16) returned -1 [10035] MyApp ERR: accept(46,
NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the control is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Note that if the specified file cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.TraceFlags Property

Gets and sets a value which specifies the network function tracing flags.

[Visual Basic]

```
Public Property TraceFlags As TraceOptions
```

[C#]

```
public InternetServer.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration which specifies the amount of detail written to the trace logfile.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Because network function tracing is enabled per-process, the trace flags are shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFlags** property should be set to the same value for each instance. Changing the trace flags for any one instance of the class will affect the logging performed for all SocketTools classes used by the application.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being written and the error **errorOperationWouldBlock** occurs, a warning is generated because the application simply needs to attempt to write the data at a later time.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.Version Property

Gets a value which returns the current version of the InternetServer class library.

[Visual Basic]
```
Public ReadOnly Property Version As String
```

[C#]
```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the InternetServer class library. This value can be used by an application for validation and debugging purposes.

See Also

InternetServer Class | SocketTools Namespace

InternetServer Methods

The methods of the **InternetServer** class are listed below. For a complete list of **InternetServer** class members, see the InternetServer Members topic.

Public Instance Methods

Abort	Overloaded. Abort the connection with a remote host.
Broadcast	Overloaded. Broadcast data to all active clients connected to the server
Cancel	Overloaded. Cancel the current blocking socket operation.
Disconnect	Overloaded. Disconnect the specified client connection from the server.
Dispose	Overloaded. Releases all resources used by InternetServer.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
FindClient	Overloaded. Return the socket handle for the client session with the specified moniker.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the InternetServer class.
Lock	Lock the server to synchronize access to shared data for all active client sessions.
Peek	Overloaded. Read data from the client and store it in a byte array, but do not remove the data from the socket buffers.
Read	Overloaded. Read data from the client socket and store it in a byte array.
ReadLine	Overloaded. Read up to a line of data from the client and return it in a string buffer.
Reject	Overloaded. Rejects a connection request from a client.
Reset	Reset the internal state of the object, resetting all properties to their default values.
Resolve	Resolves a host name to a host IP address.
Restart	Restarts the server and terminates all active client connections.

Resume	Resume accepting new client connections.
Start	Overloaded. Start listening for client connections on the specified IP address and port number.
Stop	Stop listening for new client connections and terminate all active clients already connected to the server.
Suspend	Overloaded. Suspend accepting new client connections with additional options.
Throttle	Overloaded. Limit the maximum number of client connections, connections per IP address and connection rate.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the server.
Unlock	Unlock the server and allow other server threads to resume execution.
Write	Overloaded. Write one or more bytes of data to a client.
WriteLine	Overloaded. Send a line of text to a client, terminated by a carriage-return and linefeed.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the InternetServer class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.Abort Method

Abort the connection with a remote host.

Overload List

Abort the connection with a remote host.

public void Abort();

Abort the connection with a remote host.

public void Abort(int);

See Also

InternetServer Class | SocketTools Namespace | Disconnect Method

InternetServer.Abort Method ()

Abort the connection with a remote host.

[Visual Basic]

```
Overloads Public Sub Abort()
```

[C#]

```
public void Abort();
```

Remarks

The **Abort** method immediately terminates the client connection, without waiting for any remaining data in the socket buffer to be written out. This method should only be used when the connection must be closed immediately. If this method is used, the client will see the connection as being terminated abnormally.

It is recommended that applications using the **Disconnect** method unless it is absolutely necessary to terminate the connection and immediately release the socket handle.

This implementation of the method can only be used within a class event handler, or a method that has been invoked from within an event handler. If you need to call the **Abort** method outside of an event handler, you must explicitly specify the client handle.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Abort Overload List | Disconnect Method

InternetServer.Abort Method (Int32)

Abort the connection with a remote host.

[Visual Basic]

```
Overloads Public Sub Abort( _
   ByVal handle As Integer _
)
```

[C#]

```
public void Abort(
   int handle
);
```

Parameters

handle

An integer value which specifies the handle to the client session.

Remarks

The **Abort** method immediately terminates the client connection, without waiting for any remaining data in the socket buffers to be written out. This method should only be used when the connection must be closed immediately. If this method is used, the client will see the connection as being terminated abnormally.

It is recommended that applications using the **Disconnect** method unless it is absolutely necessary to terminate the connection and immediately release the socket handle.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Abort Overload List | Disconnect Method

InternetServer.Broadcast Method

Broadcast data to all active clients connected to the server

Overload List

Broadcast data to all active clients connected to the server

public int Broadcast(byte[]);

Broadcast data to all active clients connected to the server

public int Broadcast(byte[],int);

Broadcast data to all active clients connected to the server

public int Broadcast(string);

Broadcast data to all active clients connected to the server

public int Broadcast(string,int);

See Also

InternetServer Class | SocketTools Namespace

InternetServer.Broadcast Method (Byte[])

Broadcast data to all active clients connected to the server

[Visual Basic]

```
Overloads Public Function Broadcast( _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Broadcast(
   byte[] buffer
);
```

Parameters

buffer

A byte array that contains the data that will be broadcast.

Return Value

An integer value which specifies the number of clients that the data was broadcast to. A return value of -1 indicates an error condition, and the value of the **LastError** property will indicate the cause of the failure.

Remarks

The **Broadcast** method broadcasts contents of the specified byte array to all clients connected to the server. If this method is called inside a server event handler, the message is broadcast to all clients except for the current, active client that is processing the event notification. If this method is called outside of an event handler, the data is broadcast to all connected clients.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Broadcast Overload List

InternetServer.Broadcast Method (Byte[], Int32)

Broadcast data to all active clients connected to the server

[Visual Basic]

```
Overloads Public Function Broadcast( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Broadcast(
   byte[] buffer,
   int length
);
```

Parameters

buffer
A byte array that contains the data that will be broadcast.

length
An integer value which specifies the maximum number of bytes of data to broadcast. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of clients that the data was broadcast to. A return value of -1 indicates an error condition, and the value of the **LastError** property will indicate the cause of the failure.

Remarks

The **Broadcast** method broadcasts contents of the specified byte array to all clients connected to the server. If this method is called inside a server event handler, the message is broadcast to all clients except for the current, active client that is processing the event notification. If this method is called outside of an event handler, the data is broadcast to all connected clients.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Broadcast Overload List

InternetServer.Broadcast Method (String)

Broadcast data to all active clients connected to the server

[Visual Basic]
```
Overloads Public Function Broadcast( _
   ByVal buffer As String _
) As Integer
```

[C#]
```
public int Broadcast(
   string buffer
);
```

Parameters

buffer
A string that contains the data that will be broadcast.

Return Value

An integer value which specifies the number of clients that the data was broadcast to. A return value of -1 indicates an error condition, and the value of the **LastError** property will indicate the cause of the failure.

Remarks

The **Broadcast** method broadcasts contents of the specified string to all clients connected to the server. If this method is called inside a server event handler, the message is broadcast to all clients except for the current, active client that is processing the event notification. If this method is called outside of an event handler, the data is broadcast to all connected clients.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Broadcast Overload List

InternetServer.Broadcast Method (String, Int32)

Broadcast data to all active clients connected to the server

[Visual Basic]

```
Overloads Public Function Broadcast( _
   ByVal buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Broadcast(
   string buffer,
   int length
);
```

Parameters

buffer

A string that contains the data that will be broadcast.

length

An integer value which specifies the maximum number of bytes of data to broadcast. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of clients that the data was broadcast to. A return value of -1 indicates an error condition, and the value of the **LastError** property will indicate the cause of the failure.

Remarks

The **Broadcast** method broadcasts contents of the specified string to all clients connected to the server. If this method is called inside a server event handler, the message is broadcast to all clients except for the current, active client that is processing the event notification. If this method is called outside of an event handler, the data is broadcast to all connected clients.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Broadcast Overload List

InternetServer.Cancel Method

Cancel the current blocking socket operation.

Overload List

Cancel the current blocking socket operation.

public void Cancel();

Cancel the current blocking socket operation.

public void Cancel(int);

See Also

InternetServer Class | SocketTools Namespace

InternetServer.Cancel Method ()

Cancel the current blocking socket operation.

[Visual Basic]

```
Overloads Public Sub Cancel()
```

[C#]

```
public void Cancel();
```

Remarks

When the **Cancel** method is called, the blocking socket operation will not immediately fail. An internal flag is set which causes the blocking operation to exit with an error. This means that the application cannot cancel an operation and immediately perform some other blocking function. Instead it must allow the calling stack to unwind, returning back to the blocking operation before making any further function calls.

This implementation of the method can only be used within a class event handler, or a method that has been invoked from within an event handler. If you need to call the **Cancel** method outside of an event handler, you must explicitly specify the client handle.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Cancel Overload List

InternetServer.Cancel Method (Int32)

Cancel the current blocking socket operation.

[Visual Basic]

```
Overloads Public Sub Cancel( _
   ByVal handle As Integer _
)
```

[C#]

```
public void Cancel(
   int handle
);
```

Parameters

handle

An integer value which specifies the handle to the client session.

Remarks

When the **Cancel** method is called, the blocking socket operation will not immediately fail. An internal flag is set which causes the blocking operation to exit with an error. This means that the application cannot cancel an operation and immediately perform some other blocking function. Instead it must allow the calling stack to unwind, returning back to the blocking operation before making any further function calls.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Cancel Overload List

InternetServer.Disconnect Method

Disconnect the specified client connection from the server.

Overload List

Disconnect the specified client connection from the server.

public void Disconnect();

Disconnect the specified client connection from the server.

public void Disconnect(int);

See Also

InternetServer Class | SocketTools Namespace

InternetServer.Disconnect Method ()

Disconnect the specified client connection from the server.

[Visual Basic]

```
Overloads Public Sub Disconnect()
```

[C#]

```
public void Disconnect();
```

Remarks

The **Disconnect** method terminates the specified client connection with the server and closes the socket handle allocated by the class. Note that the client socket is not immediately released when the connection is terminated and will enter a wait state for two minutes. After the time wait period has elapsed, the socket will be released by the operating system. This is a normal safety mechanism to handle any packets that may arrive after the connection has been closed.

This implementation of the method can only be used within a class event handler, or a method that has been invoked from within an event handler. If you need to call the **Disconnect** method outside of an event handler, you must explicitly specify the client handle.

To immediately terminate the connection and release the socket, use the **Abort** method.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Disconnect Overload List

InternetServer.Disconnect Method (Int32)

Disconnect the specified client connection from the server.

[Visual Basic]

```
Overloads Public Sub Disconnect( _
   ByVal handle As Integer _
)
```

[C#]

```
public void Disconnect(
   int handle
);
```

Parameters

handle

An integer value which specifies the handle to the client session.

Remarks

The **Disconnect** method terminates the specified client connection with the server and closes the socket handle allocated by the class. Note that the client socket is not immediately released when the connection is terminated and will enter a wait state for two minutes. After the time wait period has elapsed, the socket will be released by the operating system. This is a normal safety mechanism to handle any packets that may arrive after the connection has been closed.

To immediately terminate the connection and release the socket, use the **Abort** method.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Disconnect Overload List | Abort Method

InternetServer.Dispose Method

Releases all resources used by InternetServer.

Overload List

Releases all resources used by InternetServer.

public void Dispose();

Releases the unmanaged resources allocated by the InternetServer class and optionally releases the managed resources.

protected void Dispose(bool);

See Also

InternetServer Class | SocketTools Namespace

InternetServer.Dispose Method ()

Releases all resources used by InternetServer.

[Visual Basic]

```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]

```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method stops the server, terminates all active client sessions and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Dispose Overload List

InternetServer.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the InternetServer class and optionally releases the managed resources.

[Visual Basic]
```
Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]
```
protected void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **InternetServer** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Dispose Overload List

InternetServer.Finalize Method

Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.

[Visual Basic]

```
Overrides Protected Sub Finalize()
```

[C#]

```
protected override void Finalize();
```

See Also

InternetServer Class | SocketTools Namespace

InternetServer.FindClient Method

Return the socket handle for the client session with the specified ID.

Overload List

Return the socket handle for the client session with the specified ID.

public int FindClient(int);

Return the socket handle for the client session with the specified moniker.

public int FindClient(string);

See Also

InternetServer Class | SocketTools Namespace | ClientId Property

InternetServer.FindClient Method (Int32)

Return the socket handle for the client session with the specified ID.

[Visual Basic]

```
Overloads Public Function FindClient( _
   ByVal clientId As Integer _
) As Integer
```

[C#]

```
public int FindClient(
   int clientId
);
```

Parameters

clientId

An integer value which uniquely identifies the client session.

Return Value

An integer value which specifies the socket handle for the client session. If the specified client ID does not match an active client session, the method will return a value of -1 and the value of the **LastError** property will indicate the cause of the failure.

Remarks

Each client connection that is accepted by the server is assigned a unique numeric value. This value can be used by the application to identify that client session, and is different than the socket handle allocated for the client. While it is possible for a client socket handle to be reused by the operating system, client IDs are unique throughout the life of the server session and are never duplicated.

The application can determine the ID assigned to the current client session using the **ClientId** property from within an event handler such as **OnConnect**.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.FindClient Overload List | ClientId Property

InternetServer.FindClient Method (String)

Return the socket handle for the client session with the specified moniker.

[Visual Basic]

```
Overloads Public Function FindClient( _
   ByVal clientName As String _
) As Integer
```

[C#]

```
public int FindClient(
   string clientName
);
```

Parameters

clientName

A string which specifies the moniker for the client session.

Return Value

An integer value which specifies the socket handle for the client session. If the specified moniker does not match an active client session, the method will return a value of -1 and the value of the **LastError** property will indicate the cause of the failure.

Remarks

A client moniker is a string which can be used to uniquely identify a specific client session aside from its socket handle. A moniker can be assigned to the client session by setting the **ClientName** property from within a class event handler such as the **OnConnect** event.

Monikers are not case-sensitive, and they must be unique so that no client socket for a particular server can have the same moniker. The maximum length for a moniker is 127 characters.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.FindClient Overload List | ClientName Property

InternetServer.Initialize Method

Initialize an instance of the InternetServer class.

Overload List

Initialize an instance of the InternetServer class.

public bool Initialize();

Initialize an instance of the InternetServer class.

public bool Initialize(string);

See Also

InternetServer Class | SocketTools Namespace | Uninitialize Method

InternetServer.Initialize Method ()

Initialize an instance of the InternetServer class.

[Visual Basic]

```
Overloads Public Function Initialize() As Boolean
```

[C#]

```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the InternetServer class, loading the networking library and allocating resources for the current thread. Typically it is not necessary to explicitly call this method because the instance of the class is initialized by the class constructor. However, if the **Uninitialize** method is called, the class must be re-initialized before any other methods are called.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Initialize Overload List | Uninitialize Method

InternetServer.Initialize Method (String)

Initialize an instance of the InternetServer class.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the InternetServer class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of InternetServer can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the InternetServer class.

Example

The following example shows how to use the Initialize method to initialize an instance of the class. This example assumes that the license key string has been defined in code.

```
SocketTools.InternetServer server = new SocketTools.InternetServer();

if (server.Initialize(strLicenseKey) == false)
{
    MessageBox.Show(server.LastErrorString, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    return;
}
```

```
Dim Server As New SocketTools.InternetServer

If Server.Initialize(strLicenseKey) = False Then
    MsgBox(Server.LastErrorString, vbIconExclamation)
    Exit Sub
End If
```

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

InternetServer.Lock Method

Lock the server to synchronize access to shared data for all active client sessions.

[Visual Basic]
```
Public Function Lock() As Boolean
```

[C#]
```
public bool Lock();
```

Return Value

A boolean value which specifies if the server was locked. A return value of **true** specifies that the server was locked, and all other threads being managed by the server have been blocked. A return value of **false** indicates that the server could not be locked, typically because a potential deadlock was detected.

Remarks

The **Lock** method causes the server to enter a locked state where only the current thread may interact with the server and the clients that are connected to it. While a server is locked, all other threads will block when they attempt to perform a network operation. When the server is unlocked, the blocked threads will resume normal execution.

This method should be used carefully, and a server should never be left in a locked state for an extended period of time. It is meant to be used when the server process updates a global data structure and it must prevent any other threads from performing a network operation during the update. Only one server can be locked at any one time, and once a server has been locked, it can only be unlocked by the same thread.

The program should always check the return value from this method, and should never assume that the lock has been established. If more than one thread attempts to lock a server at the same time, there is no guarantee as to which thread will actually establish the lock. If a potential deadlock situation is detected, this function will fail and return a value of false.

Every time the **Lock** method is called, an internal lock counter is incremented, and the lock will not be released until the lock count drops to zero. This means that each call to the **Lock** method must be matched by an equal number of calls to the **Unlock** method. Failure to do so will result in the server becoming non-responsive as it remains in a locked state.

The **IsLocked** property can be used to determine if the server has been locked.

See Also

InternetServer Class | SocketTools Namespace | Unlock Method | IsLocked Property

InternetServer.Peek Method

Return the number of bytes available to be read from the client socket.

Overload List

Return the number of bytes available to be read from the client socket.

public int Peek();

Read data from the client and store it in a byte array, but do not remove the data from the socket buffers.

public int Peek(byte[]);

Read data from the client and store it in a byte array, but do not remove the data from the socket buffers.

public int Peek(byte[],int);

Return the number of bytes available to be read from the client socket.

public int Peek(int);

Read data from the client and store it in a byte array, but do not remove the data from the socket buffers.

public int Peek(int,byte[]);

Read data from the client and store it in a byte array, but do not remove the data from the socket buffers.

public int Peek(int,byte[],int);

Read data from the client and store it in a string, but do not remove the data from the socket buffers.

public int Peek(int,ref string);

Read data from the client and store it in a string, but do not remove the data from the socket buffers.

public int Peek(int,ref string,int);

Read data from the client and store it in a string, but do not remove the data from the socket buffers.

public int Peek(ref string);

Read data from the client and store it in a string, but do not remove the data from the socket buffers.

public int Peek(ref string,int);

See Also

InternetServer Class | SocketTools Namespace | IsReadable Property | OnRead Event

InternetServer.Peek Method ()

Return the number of bytes available to be read from the client socket.

[Visual Basic]
```
Overloads Public Function Peek() As Integer
```

[C#]
```
public int Peek();
```

Return Value

An integer value which specifies the number of bytes available to be read from the client socket. A return value of zero specifies that there is no data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Peek** method returns the number of bytes that can be read in a single operation. However, it is important to note that it may not indicate the total amount of data available to be read from the socket at that time.

If no data is available to be read, the method will return a value of zero. Using this method in a loop to poll a socket can cause the application to become non-responsive. To determine if there is data available to be read, use the **IsReadable** property.

This implementation of the method can only be used within a class event handler, or a method that has been invoked from within an event handler. If you need to call the **Peek** method outside of an event handler, you must explicitly specify the client handle.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Peek Overload List | IsReadable Property | OnRead Event

InternetServer.Peek Method (Byte[])

Read data from the client and store it in a byte array, but do not remove the data from the socket buffers.

[Visual Basic]

```
Overloads Public Function Peek( _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Peek(
   byte[] buffer
);
```

Parameters

buffer

A byte array that the data will be stored in.

Return Value

An integer value which specifies the number of bytes actually read from the socket. A return value of zero specifies that there is no data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Peek** method returns data that is available to read from the client. The data returned by this method is not removed from the socket buffers. It must be consumed by a subsequent call to the **Read** method. The return value indicates the number of bytes that can be read in a single operation. However, it is important to note that it may not indicate the total amount of data available to be read from the socket at that time.

If no data is available to be read, the method will return a value of zero. To determine if there is data available to be read, use the **IsReadable** property.

This implementation of the method can only be used within a class event handler, or a method that has been invoked from within an event handler. If you need to call the **Peek** method outside of an event handler, you must explicitly specify the client handle.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Peek Overload List | Read Method | IsReadable Property | OnRead Event

InternetServer.Peek Method (Byte[], Int32)

Read data from the client and store it in a byte array, but do not remove the data from the socket buffers.

[Visual Basic]

```
Overloads Public Function Peek( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Peek(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array that the data will be stored in.

length

An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually read from the client. A return value of zero specifies that there is no data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Peek** method returns data that is available to read from the socket, up to the number of bytes specified. The data returned by this method is not removed from the socket buffers. It must be consumed by a subsequent call to the **Read** method. The return value indicates the number of bytes that can be read in a single operation. However, it is important to note that it may not indicate the total amount of data available to be read from the socket at that time.

If no data is available to be read, the method will return a value of zero. To determine if there is data available to be read, use the **IsReadable** property.

This implementation of the method can only be used within a class event handler, or a method that has been invoked from within an event handler. If you need to call the **Peek** method outside of an event handler, you must explicitly specify the client handle.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Peek Overload List | Read Method | IsReadable Property | OnRead Event

InternetServer.Peek Method (Int32)

Return the number of bytes available to be read from the client socket.

[Visual Basic]

```
Overloads Public Function Peek( _
   ByVal handle As Integer _
) As Integer
```

[C#]

```
public int Peek(
   int handle
);
```

Parameters

handle

An integer value which specifies the handle to the client session.

Return Value

An integer value which specifies the number of bytes available to be read from the specified client socket. A return value of zero specifies that there is no data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Peek** method returns the number of bytes that can be read in a single operation. However, it is important to note that it may not indicate the total amount of data available to be read from the socket at that time.

If no data is available to be read, the method will return a value of zero. Using this method in a loop to poll a socket can cause the application to become non-responsive. To determine if there is data available to be read, use the **IsReadable** property.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Peek Overload List | OnRead Event

InternetServer.Peek Method (Int32, Byte[])

Read data from the client and store it in a byte array, but do not remove the data from the socket buffers.

[Visual Basic]

```
Overloads Public Function Peek( _
   ByVal handle As Integer, _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Peek(
   int handle,
   byte[] buffer
);
```

Parameters

handle

An integer value which specifies the handle to the client session.

buffer

A byte array that the data will be stored in.

Return Value

An integer value which specifies the number of bytes actually read from the socket. A return value of zero specifies that there is no data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Peek** method returns data that is available to read from the client, up to the number of bytes specified. The data returned by this method is not removed from the socket buffers. It must be consumed by a subsequent call to the **Read** method. The return value indicates the number of bytes that can be read in a single operation. However, it is important to note that it may not indicate the total amount of data available to be read from the socket at that time.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Peek Overload List | Read Method | OnRead Event

InternetServer.Peek Method (Int32, Byte[], Int32)

Read data from the client and store it in a byte array, but do not remove the data from the socket buffers.

[Visual Basic]
```
Overloads Public Function Peek( _
   ByVal handle As Integer, _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]
```
public int Peek(
   int handle,
   byte[] buffer,
   int length
);
```

Parameters

handle
An integer value which specifies the handle to the client session.

buffer
A byte array that the data will be stored in.

length
An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually read from the socket. A return value of zero specifies that there is no data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Peek** method returns data that is available to read from the client, up to the number of bytes specified. The data returned by this method is not removed from the socket buffers. It must be consumed by a subsequent call to the **Read** method. The return value indicates the number of bytes that can be read in a single operation. However, it is important to note that it may not indicate the total amount of data available to be read from the socket at that time.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Peek Overload List | Read Method | OnRead Event

InternetServer.Peek Method (Int32, String)

Read data from the client and store it in a string, but do not remove the data from the socket buffers.

[Visual Basic]

```
Overloads Public Function Peek( _
   ByVal handle As Integer, _
   ByRef buffer As String _
) As Integer
```

[C#]

```
public int Peek(
   int handle,
   ref string buffer
);
```

Parameters

handle

An integer value which specifies the handle to the client session.

buffer

A string that will contain the data read from the socket.

Return Value

An integer value which specifies the number of bytes actually read from the socket. A return value of zero specifies that there is no data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Peek** method returns data that is available to read from the socket, up to a maximum of 8192 bytes. The data returned by this method is not removed from the socket buffers. It must be consumed by a subsequent call to the **Read** method. The return value indicates the number of bytes that can be read in a single operation. However, it is important to note that it may not indicate the total amount of data available to be read from the socket at that time.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Peek Overload List | Read Method | OnRead Event

InternetServer.Peek Method (Int32, String, Int32)

Read data from the client and store it in a string, but do not remove the data from the socket buffers.

[Visual Basic]

```
Overloads Public Function Peek( _
   ByVal handle As Integer, _
   ByRef buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Peek(
   int handle,
   ref string buffer,
   int length
);
```

Parameters

handle

An integer value which specifies the handle to the client session.

buffer

A string that will contain the data read from the client.

length

An integer value which specifies the maximum number of bytes of data to read.

Return Value

An integer value which specifies the number of bytes actually read from the client. A return value of zero specifies that there is no data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Peek** method returns data that is available to read from the client, up to the number of bytes specified. The data returned by this method is not removed from the socket buffers. It must be consumed by a subsequent call to the **Read** method. The return value indicates the number of bytes that can be read in a single operation. However, it is important to note that it may not indicate the total amount of data available to be read from the socket at that time.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Peek Overload List | Read Method | OnRead Event

InternetServer.Peek Method (String)

Read data from the client and store it in a string, but do not remove the data from the socket buffers.

[Visual Basic]

```
Overloads Public Function Peek( _
   ByRef buffer As String _
) As Integer
```

[C#]

```
public int Peek(
   ref string buffer
);
```

Parameters

buffer

A string that will contain the data read from the socket.

Return Value

An integer value which specifies the number of bytes actually read from the client. A return value of zero specifies that there is no data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Peek** method returns data that is available to read from the client, up to a maximum of 8192 bytes. The data returned by this method is not removed from the socket buffers. It must be consumed by a subsequent call to the **Read** method. The return value indicates the number of bytes that can be read in a single operation. However, it is important to note that it may not indicate the total amount of data available to be read from the socket at that time.

If no data is available to be read, the method will return a value of zero. To determine if there is data available to be read, use the **IsReadable** property.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

This implementation of the method can only be used within a class event handler, or a method that has been invoked from within an event handler. If you need to call the **Peek** method outside of an event handler, you must explicitly specify the client handle.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Peek Overload List | Read Method | IsReadable Property | OnRead Event

InternetServer.Peek Method (String, Int32)

Read data from the client and store it in a string, but do not remove the data from the socket buffers.

[Visual Basic]

```
Overloads Public Function Peek( _
   ByRef buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Peek(
   ref string buffer,
   int length
);
```

Parameters

buffer

A string that will contain the data read from the client.

length

An integer value which specifies the maximum number of bytes of data to read.

Return Value

An integer value which specifies the number of bytes actually read from the client. A return value of zero specifies that there is no data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Peek** method returns data that is available to read from the socket, up to the number of bytes specified. The data returned by this method is not removed from the socket buffers. It must be consumed by a subsequent call to the **Read** method. The return value indicates the number of bytes that can be read in a single operation. However, it is important to note that it may not indicate the total amount of data available to be read from the socket at that time.

If no data is available to be read, the method will return a value of zero. To determine if there is data available to be read, use the **IsReadable** property.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

This implementation of the method can only be used within a class event handler, or a method that has been invoked from within an event handler. If you need to call the **Peek** method outside of an event handler, you must explicitly specify the client handle.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Peek Overload List | Read Method | IsReadable Property | OnRead Event

InternetServer.Read Method

Read data from the client socket and store it in a byte array.

Overload List

Read data from the client socket and store it in a byte array.

public int Read(byte[]);

Read data from the client socket and store it in a byte array.

public int Read(byte[],int);

Read data from the client socket and store it in a byte array.

public int Read(int,byte[]);

Read data from the client socket and store it in a byte array.

public int Read(int,byte[],int);

Read data from the client socket and store it in a string.

public int Read(int,ref string);

Read data from the client socket and store it in a string.

public int Read(int,ref string,int);

Read data from the client socket and store it in a string.

public int Read(ref string);

Read data from the client socket and store it in a string.

public int Read(ref string,int);

See Also

InternetServer Class | SocketTools Namespace

InternetServer.Read Method (Byte[])

Read data from the client socket and store it in a byte array.

[Visual Basic]

```
Overloads Public Function Read( _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Read(
   byte[] buffer
);
```

Parameters

buffer

A byte array that the data will be stored in.

Return Value

An integer value which specifies the number of bytes actually read from the client. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the socket, up to the size of the byte array passed to the method. If no data is available to be read, the calling thread will block until data is received from the server or the connection is closed.

This implementation of the method can only be used within a class event handler, or a method that has been invoked from within an event handler. If you need to call the **Read** method outside of an event handler, you must explicitly specify the client handle.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Read Overload List

InternetServer.Read Method (Byte[], Int32)

Read data from the client socket and store it in a byte array.

[Visual Basic]

```
Overloads Public Function Read( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Read(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array that the data will be stored in.

length

An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually read from the client. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the socket, up to the number of bytes specified. If no data is available to be read, the calling thread will block until data is received from the server or the connection is closed.

This implementation of the method can only be used within a class event handler, or a method that has been invoked from within an event handler. If you need to call the **Read** method outside of an event handler, you must explicitly specify the client handle.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Read Overload List

InternetServer.Read Method (Int32, Byte[])

Read data from the client socket and store it in a byte array.

[Visual Basic]

```
Overloads Public Function Read( _
   ByVal handle As Integer, _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Read(
   int handle,
   byte[] buffer
);
```

Parameters

handle

An integer value which specifies the handle to the client session.

buffer

A byte array that the data will be stored in.

Return Value

An integer value which specifies the number of bytes actually read from the specified client socket. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the socket, up to the size of the byte array passed to the method. If no data is available to be read, the calling thread will block until data is received from the server or the connection is closed.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Read Overload List

InternetServer.Read Method (Int32, Byte[], Int32)

Read data from the client socket and store it in a byte array.

[Visual Basic]

```
Overloads Public Function Read( _
   ByVal handle As Integer, _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Read(
   int handle,
   byte[] buffer,
   int length
);
```

Parameters

handle

An integer value which specifies the handle to the client session.

buffer

A byte array that the data will be stored in.

length

An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually read from the specified client socket. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the socket, up to the number of bytes specified. If no data is available to be read, the calling thread will block until data is received from the server or the connection is closed.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Read Overload List

InternetServer.Read Method (Int32, String)

Read data from the client socket and store it in a string.

[Visual Basic]

```
Overloads Public Function Read( _
   ByVal handle As Integer, _
   ByRef buffer As String _
) As Integer
```

[C#]

```
public int Read(
   int handle,
   ref string buffer
);
```

Parameters

handle

An integer value which specifies the handle to the client session.

buffer

A string that will contain the data read from the socket.

Return Value

An integer value which specifies the number of bytes actually read from the specified client socket. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the socket, up to a maximum of 8192 bytes. If no data is available to be read, the calling thread will block until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Read Overload List

InternetServer.Read Method (Int32, String, Int32)

Read data from the client socket and store it in a string.

[Visual Basic]

```
Overloads Public Function Read( _
   ByVal handle As Integer, _
   ByRef buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Read(
   int handle,
   ref string buffer,
   int length
);
```

Parameters

handle

An integer value which specifies the handle to the client session.

buffer

A string that will contain the data read from the socket.

length

An integer value which specifies the maximum number of bytes of data to read.

Return Value

An integer value which specifies the number of bytes actually read from the specified client socket. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the specified client socket, up to the number of bytes specified. If no data is available to be read, the calling thread will block until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Read Overload List

InternetServer.Read Method (String)

Read data from the client socket and store it in a string.

[Visual Basic]

```
Overloads Public Function Read( _
   ByRef buffer As String _
) As Integer
```

[C#]

```
public int Read(
   ref string buffer
);
```

Parameters

buffer

A string that will contain the data read from the client.

Return Value

An integer value which specifies the number of bytes actually read from the client. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the client, up to a maximum of 8192 bytes. If no data is available to be read, the calling thread will block until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

This implementation of the method can only be used within a class event handler, or a method that has been invoked from within an event handler. If you need to call the **Read** method outside of an event handler, you must explicitly specify the client handle.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Read Overload List

InternetServer.Read Method (String, Int32)

Read data from the client socket and store it in a string.

[Visual Basic]

```
Overloads Public Function Read( _
   ByRef buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Read(
   ref string buffer,
   int length
);
```

Parameters

buffer

A string that will contain the data read from the client.

length

An integer value which specifies the maximum number of bytes of data to read.

Return Value

An integer value which specifies the number of bytes actually read from the client. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the client, up to the number of bytes specified. If no data is available to be read, the calling thread will block until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

This implementation of the method can only be used within a class event handler, or a method that has been invoked from within an event handler. If you need to call the **Read** method outside of an event handler, you must explicitly specify the client handle.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Read Overload List

InternetServer.ReadLine Method

Read up to a line of data from the client and return it in a string buffer.

Overload List

Read up to a line of data from the client and return it in a string buffer.

public bool ReadLine(int,ref string);

Read up to a line of data from the client and return it in a string buffer.

public bool ReadLine(int,ref string,int);

Read up to a line of data from the client and return it in a string buffer.

public bool ReadLine(ref string);

Read up to a line of data from the client and return it in a string buffer.

public bool ReadLine(ref string,int);

See Also

InternetServer Class | SocketTools Namespace

InternetServer.ReadLine Method (Int32, String)

Read up to a line of data from the client and return it in a string buffer.

[Visual Basic]

```
Overloads Public Function ReadLine( _
   ByVal handle As Integer, _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool ReadLine(
   int handle,
   ref string buffer
);
```

Parameters

handle

An integer value which specifies the handle to the client session.

buffer

A string which will contain the data read from the client.

Return Value

This method returns a Boolean value which specifies if a line of data has been read. A value of **true** indicates a line of data has been read. If an error occurs or there is no more data available to read, then the method will return **false**. It is possible for data to be returned in the string buffer even if the return value is **false**. Applications should check the length of the string after the method returns to determine if any data was copied into the buffer. For example, if a timeout occurs while the method is waiting for more data to arrive on the socket, it will return zero; however, data may have already been copied into the string buffer prior to the error condition. It is the responsibility of the application to process that data, regardless of the method return value.

Remarks

The **ReadLine** method reads data from the client until an end-of-line character sequence is encountered. Unlike the **Read** method which reads arbitrary bytes of data, this method is specifically designed to return a single line of text data in a string variable. When an end-of-line character sequence is encountered, the method will stop and return the data up to that point; the string will not contain the carriage-return or linefeed characters.

There are some limitations when using the **ReadLine** method. The method should only be used to read text, never binary data. In particular, it will discard nulls, linefeed and carriage return control characters. This method will force the calling thread to block until an end-of-line character sequence is processed, the read operation times out or the remote host closes its end of the socket connection.

The **Read** and **ReadLine** methods can be intermixed, however be aware that the **Read** method will consume any data that has already been buffered by the **ReadLine** method and this may have unexpected results.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.ReadLine Overload List

InternetServer.ReadLine Method (Int32, String, Int32)

Read up to a line of data from the client and return it in a string buffer.

[Visual Basic]

```
Overloads Public Function ReadLine( _
   ByVal handle As Integer, _
   ByRef buffer As String, _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool ReadLine(
   int handle,
   ref string buffer,
   int length
);
```

Parameters

handle

An integer value which specifies the handle to the client session.

buffer

A string which will contain the data read from the client.

length

An integer value which specifies the maximum number of bytes of data to read.

Return Value

This method returns a Boolean value which specifies if a line of data has been read. A value of **true** indicates a line of data has been read. If an error occurs or there is no more data available to read, then the method will return **false**. It is possible for data to be returned in the string buffer even if the return value is **false**. Applications should check the length of the string after the method returns to determine if any data was copied into the buffer. For example, if a timeout occurs while the method is waiting for more data to arrive on the socket, it will return zero; however, data may have already been copied into the string buffer prior to the error condition. It is the responsibility of the application to process that data, regardless of the method return value.

Remarks

The **ReadLine** method reads data from the client up to the specified number of bytes or until an end-of-line character sequence is encountered. Unlike the **Read** method which reads arbitrary bytes of data, this method is specifically designed to return a single line of text data in a string variable. When an end-of-line character sequence is encountered, the method will stop and return the data up to that point; the string will not contain the carriage-return or linefeed characters.

There are some limitations when using the **ReadLine** method. The method should only be used to read text, never binary data. In particular, it will discard nulls, linefeed and carriage return control characters. This method will force the calling thread to block until an end-of-line character sequence is processed, the read operation times out or the remote host closes its end of the socket connection.

The **Read** and **ReadLine** methods can be intermixed, however be aware that the **Read** method will consume any data that has already been buffered by the **ReadLine** method and this may have unexpected results.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.ReadLine Overload List

InternetServer.ReadLine Method (String)

Read up to a line of data from the client and return it in a string buffer.

[Visual Basic]

```
Overloads Public Function ReadLine( _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool ReadLine(
   ref string buffer
);
```

Parameters

buffer

A string which will contain the data read from the client.

Return Value

This method returns a Boolean value which specifies if a line of data has been read. A value of **true** indicates a line of data has been read. If an error occurs or there is no more data available to read, then the method will return **false**. It is possible for data to be returned in the string buffer even if the return value is **false**. Applications should check the length of the string after the method returns to determine if any data was copied into the buffer. For example, if a timeout occurs while the method is waiting for more data to arrive on the socket, it will return zero; however, data may have already been copied into the string buffer prior to the error condition. It is the responsibility of the application to process that data, regardless of the method return value.

Remarks

The **ReadLine** method reads data from the client until an end-of-line character sequence is encountered. Unlike the **Read** method which reads arbitrary bytes of data, this method is specifically designed to return a single line of text data in a string variable. When an end-of-line character sequence is encountered, the method will stop and return the data up to that point; the string will not contain the carriage-return or linefeed characters.

There are some limitations when using the **ReadLine** method. The method should only be used to read text, never binary data. In particular, it will discard nulls, linefeed and carriage return control characters. This method will force the calling thread to block until an end-of-line character sequence is processed, the read operation times out or the remote host closes its end of the socket connection.

The **Read** and **ReadLine** methods can be intermixed, however be aware that the **Read** method will consume any data that has already been buffered by the **ReadLine** method and this may have unexpected results.

This implementation of the method can only be used within a class event handler, or a method that has been invoked from within an event handler. If you need to call the **ReadLine** method outside of an event handler, you must explicitly specify the client handle.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.ReadLine Overload List

InternetServer.ReadLine Method (String, Int32)

Read up to a line of data from the client and return it in a string buffer.

[Visual Basic]

```
Overloads Public Function ReadLine( _
   ByRef buffer As String, _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool ReadLine(
   ref string buffer,
   int length
);
```

Parameters

buffer

A string which will contain the data read from the client.

length

An integer value which specifies the maximum number of bytes of data to read.

Return Value

This method returns a Boolean value which specifies if a line of data has been read. A value of **true** indicates a line of data has been read. If an error occurs or there is no more data available to read, then the method will return **false**. It is possible for data to be returned in the string buffer even if the return value is **false**. Applications should check the length of the string after the method returns to determine if any data was copied into the buffer. For example, if a timeout occurs while the method is waiting for more data to arrive on the socket, it will return zero; however, data may have already been copied into the string buffer prior to the error condition. It is the responsibility of the application to process that data, regardless of the method return value.

Remarks

The **ReadLine** method reads data from the client up to the specified number of bytes or until an end-of-line character sequence is encountered. Unlike the **Read** method which reads arbitrary bytes of data, this method is specifically designed to return a single line of text data in a string variable. When an end-of-line character sequence is encountered, the method will stop and return the data up to that point; the string will not contain the carriage-return or linefeed characters.

There are some limitations when using the **ReadLine** method. The method should only be used to read text, never binary data. In particular, it will discard nulls, linefeed and carriage return control characters. This method will force the calling thread to block until an end-of-line character sequence is processed, the read operation times out or the remote host closes its end of the socket connection.

The **Read** and **ReadLine** methods can be intermixed, however be aware that the **Read** method will consume any data that has already been buffered by the **ReadLine** method and this may have unexpected results.

This implementation of the method can only be used within a class event handler, or a method that has been invoked from within an event handler. If you need to call the **ReadLine** method outside of an event handler, you must explicitly specify the client handle.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.ReadLine Overload List

InternetServer.Reject Method

Rejects a connection request from a client.

Overload List

Rejects a connection request from a client.

public bool Reject();

Rejects a connection request from a client.

public bool Reject(int);

See Also

InternetServer Class | SocketTools Namespace

InternetServer.Reject Method ()

Rejects a connection request from a client.

[Visual Basic]
```
Overloads Public Function Reject() As Boolean
```

[C#]
```
public bool Reject();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Reject** method rejects a pending client connection and the remote host will see this as the connection being aborted. If there are no pending client connections at the time, this method will immediately return with an error indicating that the operation would cause the thread to block.

This method is typically called from within the **OnAccept** event handler when the application determines that it does not wish to accept the incoming client connection.

This implementation of the method can only be used within a class event handler, or a method that has been invoked from within an event handler. If you need to call the **Reject** method outside of an event handler, you must explicitly specify the client handle.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Reject Overload List

InternetServer.Reject Method (Int32)

Rejects a connection request from a client.

[Visual Basic]

```
Overloads Public Function Reject( _
   ByVal handle As Integer _
) As Boolean
```

[C#]

```
public bool Reject(
   int handle
);
```

Parameters

handle

An integer value which specifies the handle to the client session.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Reject** method rejects a pending client connection and the remote host will see this as the connection being aborted. If there are no pending client connections at the time, this method will immediately return with an error indicating that the operation would cause the thread to block.

This method is typically called from within the **OnAccept** event handler when the application determines that it does not wish to accept the incoming client connection.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Reject Overload List

InternetServer.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. If a server has been started, it will be stopped and any active client connections will be terminated. All properties will be reset to their default values.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.Resolve Method

Resolves a host name to a host IP address.

[Visual Basic]

```
Public Function Resolve( _
   ByVal hostName As String, _
   ByRef hostAddress As String _
) As Boolean
```

[C#]

```
public bool Resolve(
   string hostName,
   ref string hostAddress
);
```

Parameters

hostName
A string which specifies the host name to be resolved.

hostAddress
A string which will contain the Internet address for the specified host.

Return Value

This method returns a Boolean value. If the host name can be resolved, the return value is **true**. If the host name cannot be resolved, the return value is **false**. To get extended error information, check the value of the **LastError** property.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.Restart Method

Restarts the server and terminates all active client connections.

[Visual Basic]
```
Public Function Restart() As Boolean
```

[C#]
```
public bool Restart();
```

Return Value

A boolean value which specifies if the server was restarted. A return value of **true** specifies that the server has been successfully restarted. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Restart** method terminates all active client connections, recreates a new listening socket bound to the same address and port number, and then resumes accepting new client connections.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.Resume Method

Resume accepting new client connections.

[Visual Basic]

```
Public Function Resume() As Boolean
```

[C#]

```
public bool Resume();
```

Return Value

A boolean value which specifies if the server has resumed accepting client connections. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Resume** method instructs the server to resume accepting new client connections. Any pending client connections that were requested while the server was suspended will be accepted.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.Start Method

Start listening for client connections.

Overload List

Start listening for client connections.

public bool Start();

Start listening for client connections on the specified port number.

public bool Start(int);

Start listening for client connections on the specified IP address and port number.

public bool Start(string,int);

Start listening for client connections on the specified IP address and port number.

public bool Start(string,int,int);

Start listening for client connections on the specified IP address and port number.

public bool Start(string,int,int,int);

Start listening for client connections on the specified IP address and port number.

public bool Start(string,int,int,int,int);

Start listening for client connections on the specified IP address and port number.

public bool Start(string,int,int,int,int,ServerOptions);

See Also

InternetServer Class | SocketTools Namespace | Backlog Property | MaxClients Property | Options Property | ServerAddress Property | ServerPort Property | Timeout Property

InternetServer.Start Method ()

Start listening for client connections.

[Visual Basic]
```
Overloads Public Function Start() As Boolean
```

[C#]
```
public bool Start();
```

Return Value

A boolean value which specifies if the server has been started. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Start** method begins listening for client connections on the default local address and port number. The server is started in its own thread and manages the client sessions independently of the calling thread.

The value of the **Backlog** property will determine the default size of the queue for incoming client connections. The value of the **MaxClients** property will determine the maximum number of clients that may connect to the server. The value of the **Options** property will determine the default options used when starting the server. The value of the **ServerAddress** and **ServerPort** properties will determine the address and port number that the server will accept client connections on. The value of the **Timeout** property will determine the default timeout period.

Example

The following example demonstrates creating an instance of the **InternetServer** class object and starting a server using the **Start** method.

```
Dim Server As SocketTools.InternetServer

Server = New SocketTools.InternetServer
Server.ServerAddress = TextBox1.Text.Trim()
Server.ServerPort = Val(TextBox2.Text)

If Server.Start() Then
    StatusBar1.Text = "The server has started listening for connections"
Else
    StatusBar1.Text = "The server could not be started"
End If
```

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Start Overload List | Backlog Property | MaxClients Property | Options Property | ServerAddress Property | ServerPort Property | Timeout Property

InternetServer.Start Method (Int32)

Start listening for client connections on the specified port number.

```
[Visual Basic]
Overloads Public Function Start( _
   ByVal localPort As Integer _
) As Boolean
```

```
[C#]
public bool Start(
   int localPort
);
```

Parameters

localPort
An integer value which specifies the port number that the server should use when listening for incoming client connections. Valid port numbers are in the range of 1 through 65535.

Return Value

A boolean value which specifies if the server has been started. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Start** method begins listening for client connections on the specified local address and port number. The server is started in its own thread and manages the client sessions independently of the calling thread.

The value of the **Backlog** property will determine the default size of the queue for incoming client connections. The value of the **MaxClients** property will determine the maximum number of clients that may connect to the server. The value of the **Options** property will determine the default options used when starting the server. The value of the **ServerAddress** property will determine the address that the server will accept client connections on. The value of the **Timeout** property will determine the default timeout period.

Example

The following example demonstrates creating an instance of the **InternetServer** class object and starting a server using the Start method.

```
Dim Server As SocketTools.InternetServer
Dim nLocalPort As Integer

nLocalPort = Val(TextBox1.Text)

If Server.Start(nLocalPort) Then
    StatusBar1.Text = "The server has started listening for connections"
Else
    StatusBar1.Text = "The server could not be started"
End If
```

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Start Overload List | Backlog Property |

MaxClients Property | Options Property | ServerAddress Property | Timeout Property

InternetServer.Start Method (String, Int32)

Start listening for client connections on the specified IP address and port number.

[Visual Basic]

```
Overloads Public Function Start( _
   ByVal localAddress As String, _
   ByVal localPort As Integer _
) As Boolean
```

[C#]

```
public bool Start(
   string localAddress,
   int localPort
);
```

Parameters

localAddress

A string value which specifies the IP address of the network adapter that the control should use when listening for connection requests. If this is an empty string or the special address "0.0.0.0" is specified, the server will listen for connection on all valid network interfaces configured for the local system.

localPort

An integer value which specifies the port number that the server should use when listening for incoming client connections. Valid port numbers are in the range of 1 through 65535.

Return Value

A boolean value which specifies if the server has been started. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Start** method begins listening for client connections on the specified local address and port number. The server is started in its own thread and manages the client sessions independently of the calling thread.

The value of the **Backlog** property will determine the default size of the queue for incoming client connections. The value of the **MaxClients** property will determine the maximum number of clients that may connect to the server. The value of the **Options** property will determine the default options used when starting the server. The value of the **Timeout** property will determine the default timeout period.

Example

The following example demonstrates creating an instance of the **InternetServer** class object and starting a server using the Start method.

```
Dim Server As SocketTools.InternetServer
Dim strLocalAddress As String
Dim nLocalPort As Integer

Server = New SocketTools.InternetServer

strLocalAddress = TextBox1.Text.Trim()
nLocalPort = Val(TextBox2.Text)

If Server.Start(strLocalAddress, nLocalPor) Then
    StatusBar1.Text = "The server has started listening for connections"
```

```
Else
    StatusBar1.Text = "The server could not be started"
End If
```

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Start Overload List | Backlog Property | MaxClients Property | Options Property | Timeout Property

InternetServer.Start Method (String, Int32, Int32)

Start listening for client connections on the specified IP address and port number.

[Visual Basic]

```
Overloads Public Function Start( _
   ByVal localAddress As String, _
   ByVal localPort As Integer, _
   ByVal maxClients As Integer _
) As Boolean
```

[C#]

```
public bool Start(
   string localAddress,
   int localPort,
   int maxClients
);
```

Parameters

localAddress
A string value which specifies the IP address of the network adapter that the control should use when listening for connection requests. If this is an empty string or the special address "0.0.0.0" is specified, the server will listen for connection on all valid network interfaces configured for the local system.

localPort
An integer value which specifies the port number that the server should use when listening for incoming client connections. Valid port numbers are in the range of 1 through 65535.

maxClients
An integer value which specifies the maximum number of clients that may connect to the server. A value of zero specifies that there is no fixed limit to the number of active client connections that may be established with the server. This value can be adjusted after the server has been created by calling the **Throttle** method.

Return Value

A boolean value which specifies if the server has been started. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Start** method begins listening for client connections on the specified local address and port number. The server is started in its own thread and manages the client sessions independently of the calling thread.

The value of the **Backlog** property will determine the default size of the queue for incoming client connections. The value of the **Options** property will determine the default options used when starting the server. The value of the **Timeout** property will determine the default timeout period.

Example

The following example demonstrates creating an instance of the **InternetServer** class object and starting a server using the Start method.

```
Dim Server As SocketTools.InternetServer
Dim strLocalAddress As String
Dim nLocalPort As Integer
Dim nMaxClients As Integer
```

```
Server = New SocketTools.InternetServer

strLocalAddress = TextBox1.Text.Trim()
nLocalPort = Val(TextBox2.Text)
nMaxClients = Val(TextBox4.Text)

If Server.Start(strLocalAddress, nLocalPort, nMaxClients) Then
    StatusBar1.Text = "The server has started listening for connections"
Else
    StatusBar1.Text = "The server could not be started"
End If
```

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Start Overload List | Backlog Property | Options Property | Timeout Property

InternetServer.Start Method (String, Int32, Int32, Int32)

Start listening for client connections on the specified IP address and port number.

[Visual Basic]

```
Overloads Public Function Start( _
   ByVal localAddress As String, _
   ByVal localPort As Integer, _
   ByVal backlog As Integer, _
   ByVal maxClients As Integer _
) As Boolean
```

[C#]

```
public bool Start(
   string localAddress,
   int localPort,
   int backlog,
   int maxClients
);
```

Parameters

localAddress

A string value which specifies the IP address of the network adapter that the control should use when listening for connection requests. If this is an empty string or the special address "0.0.0.0" is specified, the server will listen for connection on all valid network interfaces configured for the local system.

localPort

An integer value which specifies the port number that the server should use when listening for incoming client connections. Valid port numbers are in the range of 1 through 65535.

backlog

An integer value which specifies the maximum size of the queue used to manage pending connections to the service. If the argument is set to value which exceeds the maximum size for the underlying service provider, it will be silently adjusted to the nearest legal value. On Windows workstations, the maximum backlog value is 5. On Windows servers, the maximum value is 200.

maxClients

An integer value which specifies the maximum number of clients that may connect to the server. A value of zero specifies that there is no fixed limit to the number of active client connections that may be established with the server. This value can be adjusted after the server has been created by calling the **Throttle** method.

Return Value

A boolean value which specifies if the server has been started. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Start** method begins listening for client connections on the specified local address and port number. The server is started in its own thread and manages the client sessions independently of the calling thread.

The value of the **Options** property will determine the default options used when starting the server. The value of the **Timeout** property will determine the default timeout period.

Example

The following example demonstrates creating an instance of the **InternetServer** class object and starting a server using the Start method.

```
Dim Server As SocketTools.InternetServer
Dim strLocalAddress As String
Dim nLocalPort As Integer
Dim nBacklog As Integer
Dim nMaxClients As Integer

Server = New SocketTools.InternetServer

strLocalAddress = TextBox1.Text.Trim()
nLocalPort = Val(TextBox2.Text)
nBacklog = Val(TextBox3.Text)
nMaxClients = Val(TextBox4.Text)

If Server.Start(strLocalAddress, nLocalPort, nBacklog, nMaxClients) Then
    StatusBar1.Text = "The server has started listening for connections"
Else
    StatusBar1.Text = "The server could not be started"
End If
```

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Start Overload List | Options Property | Timeout Property

InternetServer.Start Method (String, Int32, Int32, Int32, Int32)

Start listening for client connections on the specified IP address and port number.

[Visual Basic]

```
Overloads Public Function Start( _
   ByVal localAddress As String, _
   ByVal localPort As Integer, _
   ByVal backlog As Integer, _
   ByVal maxClients As Integer, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Start(
   string localAddress,
   int localPort,
   int backlog,
   int maxClients,
   int timeout
);
```

Parameters

localAddress

A string value which specifies the IP address of the network adapter that the control should use when listening for connection requests. If this is an empty string or the special address "0.0.0.0" is specified, the server will listen for connection on all valid network interfaces configured for the local system.

localPort

An integer value which specifies the port number that the server should use when listening for incoming client connections. Valid port numbers are in the range of 1 through 65535.

backlog

An integer value which specifies the maximum size of the queue used to manage pending connections to the service. If the argument is set to value which exceeds the maximum size for the underlying service provider, it will be silently adjusted to the nearest legal value. On Windows workstations, the maximum backlog value is 5. On Windows servers, the maximum value is 200.

maxClients

An integer value which specifies the maximum number of clients that may connect to the server. A value of zero specifies that there is no fixed limit to the number of active client connections that may be established with the server. This value can be adjusted after the server has been created by calling the **Throttle** method.

timeout

An integer value which specifies the number of seconds the control will wait for a network operation to complete. The default timeout period of 20 seconds is sufficient for most applications.

Return Value

A boolean value which specifies if the server has been started. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Start** method begins listening for client connections on the specified local address and port number.

The server is started in its own thread and manages the client sessions independently of the calling thread.

The value of the **Options** property determines the default options that will be used when starting the server.

Example

The following example demonstrates creating an instance of the **InternetServer** class object and starting a server using the Start method.

```
Dim Server As SocketTools.InternetServer
Dim strLocalAddress As String
Dim nLocalPort As Integer
Dim nBacklog As Integer
Dim nMaxClients As Integer
Dim nTimeout As Integer

Server = New SocketTools.InternetServer

strLocalAddress = TextBox1.Text.Trim()
nLocalPort = Val(TextBox2.Text)
nBacklog = Val(TextBox3.Text)
nMaxClients = Val(TextBox4.Text)
nTimeout = Val(TextBox5.Text)

If Server.Start(strLocalAddress, nLocalPort, nBacklog, nMaxClients, nTimeout) Then
    StatusBar1.Text = "The server has started listening for connections"
Else
    StatusBar1.Text = "The server could not be started"
End If
```

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Start Overload List | Options Property

InternetServer.Start Method (String, Int32, Int32, Int32, Int32, ServerOptions)

Start listening for client connections on the specified IP address and port number.

[Visual Basic]
```
Overloads Public Function Start( _
   ByVal localAddress As String, _
   ByVal localPort As Integer, _
   ByVal backlog As Integer, _
   ByVal maxClients As Integer, _
   ByVal timeout As Integer, _
   ByVal options As ServerOptions _
) As Boolean
```

[C#]
```
public bool Start(
   string localAddress,
   int localPort,
   int backlog,
   int maxClients,
   int timeout,
   ServerOptions options
);
```

Parameters

localAddress

A string value which specifies the IP address of the network adapter that the control should use when listening for connection requests. If this is an empty string or the special address "0.0.0.0" is specified, the server will listen for connection on all valid network interfaces configured for the local system.

localPort

An integer value which specifies the port number that the server should use when listening for incoming client connections. Valid port numbers are in the range of 1 through 65535.

backlog

An integer value which specifies the maximum size of the queue used to manage pending connections to the service. If the argument is set to value which exceeds the maximum size for the underlying service provider, it will be silently adjusted to the nearest legal value. On Windows workstations, the maximum backlog value is 5. On Windows servers, the maximum value is 200.

maxClients

An integer value which specifies the maximum number of clients that may connect to the server. A value of zero specifies that there is no fixed limit to the number of active client connections that may be established with the server. This value can be adjusted after the server has been created by calling the **Throttle** method.

timeout

An integer value which specifies the number of seconds the control will wait for a network operation to complete. The default timeout period of 20 seconds is sufficient for most applications.

options

One or more of the ServerOptions enumeration flags.

Return Value

A boolean value which specifies if the server has been started. A return value of **true** specifies that the operation was successful. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Start** method begins listening for client connections on the specified local address and port number. The server is started in its own thread and manages the client sessions independently of the calling thread.

Example

The following example demonstrates creating an instance of the **InternetServer** class object and starting a server using the Start method.

```
Dim Server As SocketTools.InternetServer
Dim strLocalAddress As String
Dim nLocalPort As Integer
Dim nBacklog As Integer
Dim nMaxClients As Integer
Dim nTimeout As Integer

Server = New SocketTools.InternetServer

strLocalAddress = TextBox1.Text.Trim()
nLocalPort = Val(TextBox2.Text)
nBacklog = Val(TextBox3.Text)
nMaxClients = Val(TextBox4.Text)
nTimeout = Val(TextBox5.Text)

If Server.Start(strLocalAddress, nLocalPort, nBacklog, nMaxClients, nTimeout) Then
    StatusBar1.Text = "The server has started listening for connections"
Else
    StatusBar1.Text = "The server could not be started"
End If
```

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Start Overload List

InternetServer.Stop Method

Stop listening for new client connections and terminate all active clients already connected to the server.

[Visual Basic]

```
Public Function Stop() As Boolean
```

[C#]

```
public bool Stop();
```

Return Value

A boolean value which specifies if the server was stopped. A return value of **true** specifies that the server has been successfully stopped. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Stop** method instructs the server to stop accepting client connections, disconnects all active client connections and terminates the thread that is managing the server session.

If this method is called when there is one or more clients connected to the server, it will signal each client thread to terminate and then wait for the server thread to terminate. As the client sessions are terminated, the **OnDisconnect** event handler will not be invoked. If you wish to ensure that all clients are disconnected normally before stopping the server, call the **Suspend** method with the **suspendDisconnect** option and then stop the server after the last client has disconnected.

After the server has been terminated, the closed listening socket will go into a TIME-WAIT state which prevents an application from reusing the same address and port number bound to that socket for a brief period of time, typically two to four minutes. This is normal behavior designed to prevent delayed or misrouted packets of data from being read by a subsequent connection. To immediately start a new server using the same local address and port number, set the **ReuseAddress** property to a value of **true**.

See Also

InternetServer Class | SocketTools Namespace | Restart Method | Resume Method | Start Method | Throttle Method

InternetServer.Suspend Method

Suspend accepting new client connections.

Overload List

Suspend accepting new client connections.

public bool Suspend();

Suspend accepting new client connections with additional options.

public bool Suspend(SuspendOptions);

See Also

InternetServer Class | SocketTools Namespace

InternetServer.Suspend Method (SuspendOptions)

Suspend accepting new client connections with additional options.

[Visual Basic]

```
Overloads Public Function Suspend( _
   ByVal options As SuspendOptions _
) As Boolean
```

[C#]

```
public bool Suspend(
   SuspendOptions options
);
```

Parameters

options
One or more of the SuspendOptions enumeration flags.

Return Value

A boolean value which specifies if the server was suspended. A return value of **true** specifies that the server has suspended accepting new client connections. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Suspend** method instructs the server to suspend accepting new client connections. By default, any incoming client connections will be queued up to the maximum backlog value specified when the server was started. To resume accepting client connections, call the **Resume** method.

If the **suspendDisconnect** option is specified, the server will signal each client to disconnect and will stop accepting new connections. The **OnDisconnect** event handler will be invoked for each client that disconnects from the server. If the **suspendWait** option is also specified, this method will wait until the last client has disconnected from the server before returning to the caller. If there are a large number of clients connected to the server, this process may cause the application to block for an extended period of time and appear to be non-responsive to the user. For this reason, you should not specify the **suspendWait** option if the method is being called from the application's main UI thread.

To perform a graceful shutdown of the server, it is recommended that you call the **Suspend** method with the **suspendReject** and **suspendDisconnect** options. This will allow each client to disconnect from the server and the server will reject any new incoming connections. After the last client has disconnected from the server, the **OnIdle** event handler will be invoked and the application can call the **Stop** method to complete the shutdown process.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Suspend Overload List | SuspendOptions Enumeration

InternetServer.Suspend Method ()

Suspend accepting new client connections.

[Visual Basic]
```
Overloads Public Function Suspend() As Boolean
```

[C#]
```
public bool Suspend();
```

Return Value

A boolean value which specifies if the server was suspended. A return value of **true** specifies that the server has suspended accepting new client connections. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Suspend** method instructs the server to suspend accepting new client connections. By default, any incoming client connections will be queued up to the maximum backlog value specified when the server was started. To resume accepting client connections, call the **Resume** method.

It is not recommended that you leave a server in a suspended state for extended periods of time. Once the connection backlog queue has filled, subsequent incoming client connections will be rejected. If you wish to suspend the server for more than a few seconds, call the overloaded version of this method and specify the **suspendReject** option. This will reject all incoming client connections to the server, rather than forcing clients to wait.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Suspend Overload List

InternetServer.Throttle Method

Limit the maximum number of client connections.

Overload List

Limit the maximum number of client connections.

public bool Throttle(int);

Limit the maximum number of client connections and connections per IP address.

public bool Throttle(int,int);

Limit the maximum number of client connections, connections per IP address and connection rate.

public bool Throttle(int,int,int);

See Also

InternetServer Class | SocketTools Namespace

InternetServer.Throttle Method (Int32)

Limit the maximum number of client connections.

[Visual Basic]

```
Overloads Public Function Throttle( _
   ByVal maxClients As Integer _
) As Boolean
```

[C#]

```
public bool Throttle(
   int maxClients
);
```

Parameters

maxClients

An integer value that specifies the maximum number of clients that may connect to the server. A value of zero specifies that there is no fixed limit to the number of client connections.

Return Value

A boolean value which specifies if the method was successful. A return value of **true** indicates success. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

If the maximum number of client connections or maximum number of connections per address is exceeded, the server will reject subsequent connection attempts until the number of active client sessions drops below the specified threshold. Note that adjusting these values lower than the current connection limits will not affect clients that have already connected to the server. For example, if the **Start** method is called with the maximum number of clients set to 100, and then the **Throttle** method is called lowering that value to 75, no existing client connections will be affected by the change. However, the server will not accept any new connections until the number of active clients drops below 75.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Throttle Overload List

InternetServer.Throttle Method (Int32, Int32)

Limit the maximum number of client connections and connections per IP address.

[Visual Basic]

```
Overloads Public Function Throttle( _
   ByVal maxClients As Integer, _
   ByVal maxClientsPerAddress As Integer _
) As Boolean
```

[C#]

```
public bool Throttle(
   int maxClients,
   int maxClientsPerAddress
);
```

Parameters

maxClients
An integer value that specifies the maximum number of clients that may connect to the server. A value of zero specifies that there is no fixed limit to the number of client connections.

maxClientsPerAddress
An integer value that specifies the maximum number of clients that may connect to the server from the same IP address. A value of zero specifies that there is no fixed limit to the number of client connections per address. By default, there is no limit on the number of client connections per address.

Return Value

A boolean value which specifies if the method was successful. A return value of **true** indicates success. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

If the maximum number of client connections or maximum number of connections per address is exceeded, the server will reject subsequent connection attempts until the number of active client sessions drops below the specified threshold. Note that adjusting these values lower than the current connection limits will not affect clients that have already connected to the server. For example, if the **Start** method is called with the maximum number of clients set to 100, and then the **Throttle** method is called lowering that value to 75, no existing client connections will be affected by the change. However, the server will not accept any new connections until the number of active clients drops below 75.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Throttle Overload List

InternetServer.Throttle Method (Int32, Int32, Int32)

Limit the maximum number of client connections, connections per IP address and connection rate.

[Visual Basic]

```
Overloads Public Function Throttle( _
   ByVal maxClients As Integer, _
   ByVal maxClientsPerAddress As Integer, _
   ByVal connectionRate As Integer _
) As Boolean
```

[C#]

```
public bool Throttle(
   int maxClients,
   int maxClientsPerAddress,
   int connectionRate
);
```

Parameters

maxClients

An integer value that specifies the maximum number of clients that may connect to the server. A value of zero specifies that there is no fixed limit to the number of client connections.

maxClientsPerAddress

An integer value that specifies the maximum number of clients that may connect to the server from the same IP address. A value of zero specifies that there is no fixed limit to the number of client connections per address. By default, there is no limit on the number of client connections per address.

connectionRate

An integer value that specifies a restriction on the rate of client connections, limiting the number of connections that will be accepted within that period of time. A value of zero specifies that there is no restriction on the rate of client connections. The higher this value, the fewer the number of connections that will be accepted within a specific period of time. By default, there is no limit on the client connection rate.

Return Value

A boolean value which specifies if the method was successful. A return value of **true** indicates success. If an error occurs, the method returns **false** and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

If the maximum number of client connections or maximum number of connections per address is exceeded, the server will reject subsequent connection attempts until the number of active client sessions drops below the specified threshold. Note that adjusting these values lower than the current connection limits will not affect clients that have already connected to the server. For example, if the **Start** method is called with the maximum number of clients set to 100, and then the **Throttle** method is called lowering that value to 75, no existing client connections will be affected by the change. However, the server will not accept any new connections until the number of active clients drops below 75.

Increasing the connection rate value will force the server to slow down the rate at which it will accept incoming client connection requests. For example, setting this parameter to a value of 1000 would limit the server to accepting one client connection every second, while a value of 250 would allow the server to accept four client connections per second. Note that significantly increasing the amount of time the server must wait to accept client connections can exceed the connection backlog queue, resulting in client connections being rejected.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Throttle Overload List

InternetServer.Uninitialize Method

Uninitialize the class library and release any resources allocated for the server.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method terminates any active connection, releases resources allocated for the server and unloads the networking library. After this method has been called, no further network operations may be performed until the class instance has been re-initialized.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the class is no longer needed.

See Also

InternetServer Class | SocketTools Namespace | Initialize Method

InternetServer.Unlock Method

Unlock the server and allow other server threads to resume execution.

[Visual Basic]
```
Public Function Unlock() As Boolean
```

[C#]
```
public bool Unlock();
```

Return Value

A boolean value which specifies if the server was unlocked. A return value of **true** specifies that the server was unlocked, and the threads being managed by the server have resumed normal execution. A return value of **false** indicates that the server could not be unlocked, typically because a potential deadlock was detected.

Remarks

The **Unlock** method releases the lock on the server and allows any blocked threads to resume execution. Only one server may be locked at any one time, and only the thread which established the lock can unlock the server.

Every time the **Lock** method is called, an internal lock counter is incremented, and the lock will not be released until the lock count drops to zero. This means that each call to the **Lock** method must be matched by an equal number of calls to the **Unlock** method. Failure to do so will result in the server becoming non-responsive as it remains in a locked state.

The program should always check the return value from this method, and should never assume that the lock has been released. If a potential deadlock situation is detected, this method will fail and return a value of false.

The **IsLocked** property can be used to determine if the server has been locked.

See Also

InternetServer Class | SocketTools Namespace | Lock Method | IsLocked Property

InternetServer.Write Method

Write one or more bytes of data to a client.

Overload List

Write one or more bytes of data to a client.

public int Write(byte[]);

Write one or more bytes of data to a client.

public int Write(byte[],int);

Write one or more bytes of data to a client.

public int Write(int,byte[]);

Write one or more bytes of data to a client.

public int Write(int,byte[],int);

Write a string of characters to a client.

public int Write(int,string);

Write a string of characters to a client.

public int Write(int,string,int);

Write a string of characters to a client.

public int Write(string);

Write a string of characters to a client.

public int Write(string,int);

See Also

InternetServer Class | SocketTools Namespace

InternetServer.Write Method (Byte[])

Write one or more bytes of data to a client.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Write(
   byte[] buffer
);
```

Parameters

buffer

A byte array that contains the data to be written to the client.

Return Value

An integer value which specifies the number of bytes actually written to the client. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to a client. If there is enough room in the client socket's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space the method will block the current thread until the data can be sent.

This implementation of the method can only be used within a class event handler, or a method that has been invoked from within an event handler. If you need to call the **Write** method outside of an event handler, you must explicitly specify the client handle.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Write Overload List

InternetServer.Write Method (Byte[], Int32)

Write one or more bytes of data to a client.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Write(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array that contains the data to be written to the client.

length

An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually written to the client. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to a client. If there is enough room in the client socket's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space the method will block the current thread until the data can be sent.

This implementation of the method can only be used within a class event handler, or a method that has been invoked from within an event handler. If you need to call the **Write** method outside of an event handler, you must explicitly specify the client handle.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Write Overload List

InternetServer.Write Method (Int32, Byte[])

Write one or more bytes of data to a client.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal handle As Integer, _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Write(
   int handle,
   byte[] buffer
);
```

Parameters

handle
: An integer value which specifies the handle to the client session.

buffer
: A byte array that contains the data to be written to the client.

Return Value

An integer value which specifies the number of bytes actually written to the client. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to the specified client. If there is enough room in the client socket's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space the method will block the current thread until the data can be sent.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Write Overload List

InternetServer.Write Method (Int32, Byte[], Int32)

Write one or more bytes of data to a client.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal handle As Integer, _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Write(
   int handle,
   byte[] buffer,
   int length
);
```

Parameters

handle
An integer value which specifies the handle to the client session.

buffer
A byte array that contains the data to be written to the client.

length
An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually written to the client. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to the specified client. If there is enough room in the client socket's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space the method will block the current thread until the data can be sent.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Write Overload List | Broadcast Method

InternetServer.Write Method (Int32, String)

Write a string of characters to a client.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal handle As Integer, _
   ByVal buffer As String _
) As Integer
```

[C#]

```
public int Write(
   int handle,
   string buffer
);
```

Parameters

handle
An integer value which specifies the handle to the client session.

buffer
A string which contains the data to be written to the client.

Return Value

An integer value which specifies the number of characters actually written to the client. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to a client. If there is enough room in the socket's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space the method will block the current thread until the data can be sent.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Write Overload List

InternetServer.Write Method (Int32, String, Int32)

Write a string of characters to a client.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal handle As Integer, _
   ByVal buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Write(
   int handle,
   string buffer,
   int length
);
```

Parameters

handle

An integer value which specifies the handle to the client session.

buffer

A string which contains the data to be written to the client.

length

An integer value which specifies the maximum number of characters to write. This value cannot be larger than the length of the string specified by the caller.

Return Value

An integer value which specifies the number of characters actually written to the client. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to a client. If there is enough room in the socket's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space the method will block the current thread until the data can be sent.

The string will be converted to an array of bytes before being written to the socket. By default, the character encoding used will be for the current locale. Depending on the contents of the string, the number of bytes written may be different than the string length specified. This is because the conversion from Unicode to a byte array may result in a multi-byte character sequence.

You should never use strings to read and write binary data. Always use byte arrays to ensure that no character conversion is performed.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Write Overload List

InternetServer.Write Method (String)

Write a string of characters to a client.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String _
) As Integer
```

[C#]

```
public int Write(
   string buffer
);
```

Parameters

buffer

A string which contains the data to be written to the client.

Return Value

An integer value which specifies the number of characters actually written to the client. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to a client. If there is enough room in the socket's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space the method will block the current thread until the data can be sent.

This implementation of the method can only be used within a class event handler, or a method that has been invoked from within an event handler. If you need to call the **Write** method outside of an event handler, you must explicitly specify the client handle.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Write Overload List

InternetServer.Write Method (String, Int32)

Write a string of characters to a client.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Write(
   string buffer,
   int length
);
```

Parameters

buffer

A string which contains the data to be written to the client.

length

An integer value which specifies the maximum number of characters to write. This value cannot be larger than the length of the string specified by the caller.

Return Value

An integer value which specifies the number of characters actually written to the client. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to a client. If there is enough room in the socket's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space the method will block the current thread until the data can be sent.

This implementation of the method can only be used within a class event handler, or a method that has been invoked from within an event handler. If you need to call the **Write** method outside of an event handler, you must explicitly specify the client handle.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.Write Overload List

InternetServer.WriteLine Method

Send an empty line of text to a client, terminated by a carriage-return and linefeed.

Overload List

Send an empty line of text to a client, terminated by a carriage-return and linefeed.

public bool WriteLine();

Send an empty line of text to a client, terminated by a carriage-return and linefeed.

public bool WriteLine(int);

Send a line of text to a client, terminated by a carriage-return and linefeed.

public bool WriteLine(int,string);

Send a line of text to a client, terminated by a carriage-return and linefeed.

public bool WriteLine(int,string,ref int);

Send a line of text to a client, terminated by a carriage-return and linefeed.

public bool WriteLine(string);

Send a line of text to a client, terminated by a carriage-return and linefeed.

public bool WriteLine(string,ref int);

See Also

InternetServer Class | SocketTools Namespace

InternetServer.WriteLine Method ()

Send an empty line of text to a client, terminated by a carriage-return and linefeed.

[Visual Basic]

```
Overloads Public Function WriteLine() As Boolean
```

[C#]

```
public bool WriteLine();
```

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

The **WriteLine** method will send an empty line of text to the client, terminated by a carriage-return and linefeed. Calling this method will force the calling thread to block until the complete line of text has been written, the write operation times out or the remote host aborts the connection.

This implementation of the method can only be used within a class event handler, or a method that has been invoked from within an event handler. If you need to call the **WriteLine** method outside of an event handler, you must explicitly specify the client handle.

The **Write** and **WriteLine** methods can be safely intermixed.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.WriteLine Overload List

InternetServer.WriteLine Method (Int32)

Send an empty line of text to a client, terminated by a carriage-return and linefeed.

[Visual Basic]

```
Overloads Public Function WriteLine( _
   ByVal handle As Integer _
) As Boolean
```

[C#]

```
public bool WriteLine(
   int handle
);
```

Parameters

handle

An integer value which specifies the handle to the client session.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

The **WriteLine** method will send an empty line of text to the specified client, terminated by a carriage-return and linefeed. Calling this method will force the calling thread to block until the complete line of text has been written, the write operation times out or the remote host aborts the connection.

The **Write** and **WriteLine** methods can be safely intermixed.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.WriteLine Overload List

InternetServer.WriteLine Method (Int32, String)

Send a line of text to a client, terminated by a carriage-return and linefeed.

[Visual Basic]

```
Overloads Public Function WriteLine( _
   ByVal handle As Integer, _
   ByVal buffer As String _
) As Boolean
```

[C#]

```
public bool WriteLine(
   int handle,
   string buffer
);
```

Parameters

handle

An integer value which specifies the handle to the client session.

buffer

A string which contains the data that will be sent to the specified client. The data will always be terminated with a carriage-return and linefeed control character sequence. If the string is empty, then a only a carriage-return and linefeed are written to the socket. Note that if the string contains a null character, any data that follows the null character will be discarded.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

The **WriteLine** method should only be used to send text, never binary data. In particular, this method will discard any data that follows a null character and will append linefeed and carriage return control characters to the data stream. Calling this method will force the current thread to block until the complete line of text has been written, the write operation times out or the remote host aborts the connection.

The **Write** and **WriteLine** methods can be safely intermixed.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.WriteLine Overload List

InternetServer.WriteLine Method (Int32, String, Int32)

Send a line of text to a client, terminated by a carriage-return and linefeed.

[Visual Basic]

```
Overloads Public Function WriteLine( _
   ByVal handle As Integer, _
   ByVal buffer As String, _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool WriteLine(
   int handle,
   string buffer,
   ref int length
);
```

Parameters

handle

An integer value which specifies the handle to the client session.

buffer

A string which contains the data that will be sent to the specified client. The data will always be terminated with a carriage-return and linefeed control character sequence. If the string is empty, then a only a carriage-return and linefeed are written to the client. Note that if the string contains a null character, any data that follows the null character will be discarded.

length

An integer value which specifies the maximum number of characters to write. This value cannot be larger than the length of the string specified by the caller.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

The **WriteLine** method should only be used to send text, never binary data. In particular, this method will discard any data that follows a null character and will append linefeed and carriage return control characters to the data stream. Calling this method will force the current thread to block until the complete line of text has been written, the write operation times out or the remote host aborts the connection.

The string will be converted to an array of bytes before being written to the socket. By default, the character encoding used will be for the current locale. Depending on the contents of the string, the number of bytes written may be different than the string length specified. This is because the conversion from Unicode to a byte array may result in a multi-byte character sequence.

You should never use strings to read and write binary data. Always use byte arrays to ensure that no character conversion is performed.

The **Write** and **WriteLine** methods can be safely intermixed.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.WriteLine Overload List

InternetServer.WriteLine Method (String)

Send a line of text to a client, terminated by a carriage-return and linefeed.

[Visual Basic]

```
Overloads Public Function WriteLine( _
   ByVal buffer As String _
) As Boolean
```

[C#]

```
public bool WriteLine(
   string buffer
);
```

Parameters

buffer

A string which contains the data that will be sent to the client. The data will always be terminated with a carriage-return and linefeed control character sequence. If the string is empty, then a only a carriage-return and linefeed are written to the socket. Note that if the string contains a null character, any data that follows the null byte will be discarded.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

The **WriteLine** method should only be used to send text, never binary data. In particular, this method will discard any data that follows a null character and will append linefeed and carriage return control characters to the data stream. Calling this method will force the current thread to block until the complete line of text has been written, the write operation times out or the remote host aborts the connection.

This implementation of the method can only be used within a class event handler, or a method that has been invoked from within an event handler. If you need to call the **WriteLine** method outside of an event handler, you must explicitly specify the client handle.

The **Write** and **WriteLine** methods can be safely intermixed.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.WriteLine Overload List

InternetServer.WriteLine Method (String, Int32)

Send a line of text to a client, terminated by a carriage-return and linefeed.

[Visual Basic]

```
Overloads Public Function WriteLine( _
   ByVal buffer As String, _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool WriteLine(
   string buffer,
   ref int length
);
```

Parameters

buffer

A string which contains the data that will be sent to the client. The data will always be terminated with a carriage-return and linefeed control character sequence. If the string is empty, then a only a carriage-return and linefeed are written to the socket. Note that if the string contains a null character, any data that follows the null byte will be discarded.

length

An integer value which specifies the maximum number of characters to write. This value cannot be larger than the length of the string specified by the caller.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

The **WriteLine** method should only be used to send text, never binary data. In particular, this method will discard any data that follows a null character and will append linefeed and carriage return control characters to the data stream. Calling this method will force the current thread to block until the complete line of text has been written, the write operation times out or the remote host aborts the connection.

This implementation of the method can only be used within a class event handler, or a method that has been invoked from within an event handler. If you need to call the **WriteLine** method outside of an event handler, you must explicitly specify the client handle.

The **Write** and **WriteLine** methods can be safely intermixed.

See Also

InternetServer Class | SocketTools Namespace | InternetServer.WriteLine Overload List

InternetServer Events

The events of the **InternetServer** class are listed below. For a complete list of **InternetServer** class members, see the InternetServer Members topic.

Public Instance Events

⚡ OnAccept	Occurs when a client attempts to establish a connection with the server.
⚡ OnCancel	Occurs when a blocking socket operation is canceled.
⚡ OnConnect	Occurs when a connection is established with the remote host.
⚡ OnDisconnect	Occurs when the remote host disconnects from the local system.
⚡ OnError	Occurs when an socket operation fails.
⚡ OnIdle	Occurs when the there are no clients connected to the server.
⚡ OnRead	Occurs when data is available to be read from the client.
⚡ OnStart	Occurs when the server starts accepting connections.
⚡ OnStop	Occurs when the server stops accepting connections.
⚡ OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
⚡ OnWrite	Occurs when data can be written to the client.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.OnAccept Event

Occurs when a client attempts to establish a connection with the server.

```
[Visual Basic]
Public Event OnAccept As OnAcceptEventHandler
```

```
[C#]
public event OnAcceptEventHandler OnAccept;
```

Event Data

The event handler receives an argument of type InternetServer.AcceptEventArgs containing data related to this event. The following **InternetServer.AcceptEventArgs** properties provide information specific to this event.

Property	Description
ClientAddress	Gets a value that specifies the Internet address of the current client session.
ClientPort	Gets a value that specifies the port number used by the current client session.
Handle	Gets a value that specifies the socket handle for the listening server.

Remarks

The **OnAccept** event occurs when a client attempts to connect to the local system. A connection is not actually established until it has been accepted by the server.

The **ClientAddress** or **ClientHost** properties may be used to determine the Internet address and host name of the remote host that is establishing the connection. To prevent the client from completing the connection, call the **Reject** method.

After the client connection has been established and the worker thread for that client session has started, the **OnConnect** event will fire.

User interface controls can only be accessed from the UI thread that created them, and attempting to update a control from another thread can result in the program becoming non-responsive or terminating abnormally. Because this event is generated in the context of the server thread, not the thread that created the class instance, you cannot directly modify a control from within this event handler. Instead, you must create a delegate and use the **Invoke** method to marshal invocations to the associated UI thread. For more information, refer to the documentation for the control.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.AcceptEventArgs Class

Provides data for the OnAccept event.

For a list of all members of this type, see InternetServer.AcceptEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.InternetServer.AcceptEventArgs

[Visual Basic]
```
Public Class InternetServer.AcceptEventArgs
    Inherits EventArgs
```

[C#]
```
public class InternetServer.AcceptEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

AcceptEventArgs specifies the socket handle for the server that should accept the incoming client connection.

The OnAccept event occurs when a remote host attempts to establish a connection with the local system.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

InternetServer.AcceptEventArgs Members | SocketTools Namespace | OnAccept Event (SocketTools.InternetServer)

InternetServer.AcceptEventArgs Members

InternetServer.AcceptEventArgs overview

Public Instance Constructors

InternetServer.AcceptEventArgs Constructor	Initializes a new instance of the InternetServer.AcceptEventArgs class.

Public Instance Properties

ClientAddress	Gets a value that specifies the Internet address of the current client session.
ClientPort	Gets a value that specifies the port number used by the current client session.
Handle	Gets a value that specifies the socket handle for the listening server.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetServer.AcceptEventArgs Class | SocketTools Namespace | OnAccept Event (SocketTools.InternetServer)

InternetServer.AcceptEventArgs Constructor

Initializes a new instance of the InternetServer.AcceptEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public InternetServer.AcceptEventArgs();
```

See Also

InternetServer.AcceptEventArgs Class | SocketTools Namespace

InternetServer.AcceptEventArgs Properties

The properties of the **InternetServer.AcceptEventArgs** class are listed below. For a complete list of **InternetServer.AcceptEventArgs** class members, see the InternetServer.AcceptEventArgs Members topic.

Public Instance Properties

ClientAddress	Gets a value that specifies the Internet address of the current client session.
ClientPort	Gets a value that specifies the port number used by the current client session.
Handle	Gets a value that specifies the socket handle for the listening server.

See Also

InternetServer.AcceptEventArgs Class | SocketTools Namespace | OnAccept Event (SocketTools.InternetServer)

InternetServer.AcceptEventArgs.ClientAddress Property

Gets a value that specifies the Internet address of the current client session.

[Visual Basic]

```
Public ReadOnly Property ClientAddress As String
```

[C#]

```
public string ClientAddress {get;}
```

Remarks

The **ClientAddress** property will return the address of the client that is requesting the connection. The server application may use this information to determine if it wishes to accept or reject the client connection.

See Also

InternetServer.AcceptEventArgs Class | SocketTools Namespace

InternetServer.AcceptEventArgs.ClientPort Property

Gets a value that specifies the port number used by the current client session.

[Visual Basic]

```
Public ReadOnly Property ClientPort As Integer
```

[C#]

```
public int ClientPort {get;}
```

Remarks

The **ClientPort** property returns the port number that the client has used when establishing a connection with the server.

See Also

InternetServer.AcceptEventArgs Class | SocketTools Namespace

InternetServer.AcceptEventArgs.Handle Property

Gets a value that specifies the socket handle for the listening server.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer value which specifies the server socket handle.

Remarks

The **Handle** property returns the socket handle for the server that generated the event. This value is used for identification purposes only and should not be used in conjunction with methods such as **Read** and **Write**, which may only be used with client handles.

See Also

InternetServer.AcceptEventArgs Class | SocketTools Namespace

InternetServer.OnCancel Event

Occurs when a blocking socket operation is canceled.

[Visual Basic]
```
Public Event OnCancel As OnCancelEventHandler
```

[C#]
```
public event OnCancelEventHandler OnCancel;
```

Event Data

The event handler receives an argument of type InternetServer.CancelEventArgs containing data related to this event. The following **InternetServer.CancelEventArgs** property provides information specific to this event.

Property	Description
Handle	Gets a value that specifies the socket handle for the client session.

Remarks

The **OnCancel** event is generated when a blocking socket operation, such as sending or receiving data, is canceled with the **Cancel** method.

User interface controls can only be accessed from the UI thread that created them, and attempting to update a control from another thread can result in the program becoming non-responsive or terminating abnormally. Because this event is generated in the context of the client thread, not the thread that created the class instance, you cannot directly modify a control from within this event handler. Instead, you must create a delegate and use the **Invoke** method to marshal invocations to the associated UI thread. For more information, refer to the documentation for the control.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.CancelEventArgs Class

Provides data for the OnCancel event.

For a list of all members of this type, see InternetServer.CancelEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.InternetServer.CancelEventArgs

[Visual Basic]

```
Public Class InternetServer.CancelEventArgs
    Inherits EventArgs
```

[C#]

```
public class InternetServer.CancelEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

CancelEventArgs specifies the socket handle for the current client session.

The OnCancel event occurs when a blocking network operation has been canceled.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

InternetServer.CancelEventArgs Members | SocketTools Namespace

InternetServer.CancelEventArgs Members

InternetServer.CancelEventArgs overview

Public Instance Constructors

InternetServer.CancelEventArgs Constructor	Initializes a new instance of the InternetServer.CancelEventArgs class.

Public Instance Properties

Handle	Gets a value that specifies the socket handle for the client session.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetServer.CancelEventArgs Class | SocketTools Namespace

InternetServer.CancelEventArgs Constructor

Initializes a new instance of the InternetServer.CancelEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public InternetServer.CancelEventArgs();
```

See Also

InternetServer.CancelEventArgs Class | SocketTools Namespace

InternetServer.CancelEventArgs Properties

The properties of the **InternetServer.CancelEventArgs** class are listed below. For a complete list of **InternetServer.CancelEventArgs** class members, see the InternetServer.CancelEventArgs Members topic.

Public Instance Properties

Handle	Gets a value that specifies the socket handle for the client session.

See Also

InternetServer.CancelEventArgs Class | SocketTools Namespace

InternetServer.CancelEventArgs.Handle Property

Gets a value that specifies the socket handle for the client session.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer value which specifies the client socket handle.

Remarks

The **Handle** property returns the socket handle for the client that generated the event. This handle can be used in conjunction with methods such as **Read** and **Write** to exchange data with the client.

See Also

InternetServer.CancelEventArgs Class | SocketTools Namespace

InternetServer.OnConnect Event

Occurs when a connection is established with the remote host.

[Visual Basic]

```
Public Event OnConnect As OnConnectEventHandler
```

[C#]

```
public event OnConnectEventHandler OnConnect;
```

Event Data

The event handler receives an argument of type InternetServer.ConnectEventArgs containing data related to this event. The following **InternetServer.ConnectEventArgs** property provides information specific to this event.

Property	Description
Handle	Gets a value that specifies the socket handle for the client session.

Remarks

The **OnConnect** event occurs when the client connection to the server has completed.

The **ClientAddress** property can be used to determine the IP address of the client which established the connection. To terminate the client connection, use the **Disconnect** method.

User interface controls can only be accessed from the UI thread that created them, and attempting to update a control from another thread can result in the program becoming non-responsive or terminating abnormally. Because this event is generated in the context of the client thread, not the thread that created the class instance, you cannot directly modify a control from within this event handler. Instead, you must create a delegate and use the **Invoke** method to marshal invocations to the associated UI thread. For more information, refer to the documentation for the control.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.ConnectEventArgs Class

Provides data for the OnConnect event.

For a list of all members of this type, see InternetServer.ConnectEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.InternetServer.ConnectEventArgs

[Visual Basic]
```
Public Class InternetServer.ConnectEventArgs
    Inherits EventArgs
```

[C#]
```
public class InternetServer.ConnectEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ConnectEventArgs specifies the socket handle for the current client session.

The OnConnect event occurs when the client connection to the server has completed. The **Handle** property specifies the handle to the client socket that was allocated for the session. This handle can be used with methods such as **Read** and **Write** to exchange information with the client.

The **ClientAddress** property can be used to determine the IP address of the client which established the connection. To terminate the client connection, use the **Disconnect** method.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

InternetServer.ConnectEventArgs Members | SocketTools Namespace | OnConnect Event (SocketTools.InternetServer)

InternetServer.ConnectEventArgs Members

InternetServer.ConnectEventArgs overview

Public Instance Constructors

InternetServer.ConnectEventArgs Constructor	Initializes a new instance of the InternetServer.ConnectEventArgs class.

Public Instance Properties

Handle	Gets a value that specifies the socket handle for the client session.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetServer.ConnectEventArgs Class | SocketTools Namespace | OnConnect Event (SocketTools.InternetServer)

InternetServer.ConnectEventArgs Constructor

Initializes a new instance of the InternetServer.ConnectEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public InternetServer.ConnectEventArgs();
```

See Also

InternetServer.ConnectEventArgs Class | SocketTools Namespace

InternetServer.ConnectEventArgs Properties

The properties of the **InternetServer.ConnectEventArgs** class are listed below. For a complete list of **InternetServer.ConnectEventArgs** class members, see the InternetServer.ConnectEventArgs Members topic.

Public Instance Properties

Handle	Gets a value that specifies the socket handle for the client session.

See Also

InternetServer.ConnectEventArgs Class | SocketTools Namespace | OnConnect Event (SocketTools.InternetServer)

InternetServer.ConnectEventArgs.Handle Property

Gets a value that specifies the socket handle for the client session.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer value which specifies the client socket handle.

Remarks

The **Handle** property returns the socket handle for the client that generated the event. This handle can be used in conjunction with methods such as **Read** and **Write** to exchange data with the client.

See Also

InternetServer.ConnectEventArgs Class | SocketTools Namespace

InternetServer.OnDisconnect Event

Occurs when the remote host disconnects from the local system.

[Visual Basic]

```
Public Event OnDisconnect As OnDisconnectEventHandler
```

[C#]

```
public event OnDisconnectEventHandler OnDisconnect;
```

Event Data

The event handler receives an argument of type InternetServer.DisconnectEventArgs containing data related to this event. The following **InternetServer.DisconnectEventArgs** property provides information specific to this event.

Property	Description
Handle	Gets a value that specifies the socket handle for the client session.

Remarks

The **OnDisconnect** event is generated when the connection is terminated by the client and there is no more data available to be read.

It is not necessary to call the **Disconnect** method inside the **OnDisconnect** event handler because the client session is already in the process of disconnecting from the server.

User interface controls can only be accessed from the UI thread that created them, and attempting to update a control from another thread can result in the program becoming non-responsive or terminating abnormally. Because this event is generated in the context of the client thread, not the thread that created the class instance, you cannot directly modify a control from within this event handler. Instead, you must create a delegate and use the **Invoke** method to marshal invocations to the associated UI thread. For more information, refer to the documentation for the control.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.DisconnectEventArgs Class

Provides data for the OnDisconnect event.

For a list of all members of this type, see InternetServer.DisconnectEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.InternetServer.DisconnectEventArgs

[Visual Basic]

```
Public Class InternetServer.DisconnectEventArgs
    Inherits EventArgs
```

[C#]

```
public class InternetServer.DisconnectEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

DisconnectEventArgs specifies the socket handle for the current client session.

The OnDisconnect event is generated when the connection is terminated by the client and there is no more data available to be read. The **Handle** property specifies the socket handle of the client session which has terminated. It is important to note that the client handle is provided for informational purposes only and the application should not attempt to read or write data using this handle from within this event handler.

It is not necessary to call the **Disconnect** method inside the **OnDisconnect** event handler because the client session is already in the process of disconnecting from the server.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

InternetServer.DisconnectEventArgs Members | SocketTools Namespace | OnDisconnect Event (SocketTools.InternetServer)

InternetServer.DisconnectEventArgs Members

InternetServer.DisconnectEventArgs overview

Public Instance Constructors

InternetServer.DisconnectEventArgs Constructor	Initializes a new instance of the InternetServer.DisconnectEventArgs class.

Public Instance Properties

Handle	Gets a value that specifies the socket handle for the client session.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetServer.DisconnectEventArgs Class | SocketTools Namespace | OnDisconnect Event (SocketTools.InternetServer)

InternetServer.DisconnectEventArgs Constructor

Initializes a new instance of the InternetServer.DisconnectEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public InternetServer.DisconnectEventArgs();
```

See Also

InternetServer.DisconnectEventArgs Class | SocketTools Namespace

InternetServer.DisconnectEventArgs Properties

The properties of the **InternetServer.DisconnectEventArgs** class are listed below. For a complete list of **InternetServer.DisconnectEventArgs** class members, see the InternetServer.DisconnectEventArgs Members topic.

Public Instance Properties

Handle	Gets a value that specifies the socket handle for the client session.

See Also

InternetServer.DisconnectEventArgs Class | SocketTools Namespace | OnDisconnect Event (SocketTools.InternetServer)

InternetServer.DisconnectEventArgs.Handle Property

Gets a value that specifies the socket handle for the client session.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer value which specifies the client socket handle.

Remarks

The **Handle** property returns the socket handle for the client that generated the event. This handle can be used in conjunction with methods such as **Read** and **Write** to exchange data with the client.

See Also

InternetServer.DisconnectEventArgs Class | SocketTools Namespace

InternetServer.OnError Event

Occurs when an socket operation fails.

[Visual Basic]

```
Public Event OnError As OnErrorEventHandler
```

[C#]

```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type InternetServer.ErrorEventArgs containing data related to this event. The following **InternetServer.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.
Handle	Gets a value that specifies the socket handle that generated the error.

Remarks

The **OnError** event occurs when a socket operation fails.

User interface controls can only be accessed from the UI thread that created them, and attempting to update a control from another thread can result in the program becoming non-responsive or terminating abnormally. Because this event may be generated in the context of the client or server thread, not the thread that created the class instance, you cannot directly modify a control from within this event handler. Instead, you must create a delegate and use the **Invoke** method to marshal invocations to the associated UI thread. For more information, refer to the documentation for the control.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see InternetServer.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.InternetServer.ErrorEventArgs

[Visual Basic]
```
Public Class InternetServer.ErrorEventArgs
    Inherits EventArgs
```

[C#]
```
public class InternetServer.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

InternetServer.ErrorEventArgs Members | SocketTools Namespace

InternetServer.ErrorEventArgs Members

InternetServer.ErrorEventArgs overview

Public Instance Constructors

InternetServer.ErrorEventArgs Constructor	Initializes a new instance of the InternetServer.ErrorEventArgs class.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.
Handle	Gets a value that specifies the socket handle that generated the error.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetServer.ErrorEventArgs Class | SocketTools Namespace

InternetServer.ErrorEventArgs Constructor

Initializes a new instance of the InternetServer.ErrorEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public InternetServer.ErrorEventArgs();
```

See Also

InternetServer.ErrorEventArgs Class | SocketTools Namespace

InternetServer.ErrorEventArgs Properties

The properties of the **InternetServer.ErrorEventArgs** class are listed below. For a complete list of **InternetServer.ErrorEventArgs** class members, see the InternetServer.ErrorEventArgs Members topic.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.
Handle	Gets a value that specifies the socket handle that generated the error.

See Also

InternetServer.ErrorEventArgs Class | SocketTools Namespace

InternetServer.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

InternetServer.ErrorEventArgs Class | SocketTools Namespace | Error Property

InternetServer.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public InternetServer.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

InternetServer.ErrorEventArgs Class | SocketTools Namespace | Description Property

InternetServer.ErrorEventArgs.Handle Property

Gets a value that specifies the socket handle that generated the error.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer value which specifies a socket handle.

Remarks

The **Handle** property returns the socket handle for the client or server that generated the event. If no server is active, then this property will return a value of -1.

See Also

InternetServer.ErrorEventArgs Class | SocketTools Namespace

InternetServer.OnRead Event

Occurs when data is available to be read from the client.

```
[Visual Basic]
Public Event OnRead As OnReadEventHandler
```

```
[C#]
public event OnReadEventHandler OnRead;
```

Event Data

The event handler receives an argument of type InternetServer.ReadEventArgs containing data related to this event. The following **InternetServer.ReadEventArgs** property provides information specific to this event.

Property	Description
Handle	Gets a value that specifies the socket handle for the client session.

Remarks

The **OnRead** event is generated when the client sends data to the server. The **Handle** event argument property specifies the handle to the client socket which can be used with the **Read** or **ReadLine** methods to read the data that was sent.

When this event fires, it guarantees that data can be read from the specified client without causing the current thread to enter a blocked state. However, calling this method multiple times inside the event handler may cause the current thread to block when there is no more data available to read. The **IsReadable** property can be used to determine if there is additional data available to be read.

User interface controls can only be accessed from the UI thread that created them, and attempting to update a control from another thread can result in the program becoming non-responsive or terminating abnormally. Because this event is generated in the context of the client thread, not the thread that created the class instance, you cannot directly modify a control from within this event handler. Instead, you must create a delegate and use the **Invoke** method to marshal invocations to the associated UI thread. For more information, refer to the documentation for the control.

Example

```
Private Sub Server_OnRead(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Socket.OnRead
    Dim strBuffer As String
    Dim nRead As Integer

    ' Read up to m_nBufferSize bytes of data from the client
    nRead = Server1.Read(e.Handle, strBuffer, m_nBufferSize)

    If nRead > 0 Then
        ' Process the data that has been read from the client
        ProcessData(strBuffer)
    End If
End Sub
```

See Also

InternetServer Class | SocketTools Namespace

InternetServer.ReadEventArgs Class

Provides data for the OnRead event.

For a list of all members of this type, see InternetServer.ReadEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.InternetServer.ReadEventArgs

[Visual Basic]

```
Public Class InternetServer.ReadEventArgs
    Inherits EventArgs
```

[C#]

```
public class InternetServer.ReadEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ReadEventArgs specifies the socket handle for the current client session.

The OnRead event is generated when the client sends data to the server. The **Handle** event argument property specifies the handle to the client socket which can be used with the **Read** or **ReadLine** methods to read the data that was sent.

When this event fires, it guarantees that data can be read from the specified client without causing the current thread to enter a blocked state. However, calling this method multiple times inside the event handler may cause the current thread to block when there is no more data available to read. The **IsReadable** property can be used to determine if there is additional data available to be read.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

InternetServer.ReadEventArgs Members | SocketTools Namespace | OnRead Event (SocketTools.InternetServer)

InternetServer.ReadEventArgs Members

InternetServer.ReadEventArgs overview

Public Instance Constructors

InternetServer.ReadEventArgs Constructor	Initializes a new instance of the InternetServer.ReadEventArgs class.

Public Instance Properties

Handle	Gets a value that specifies the socket handle for the client session.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetServer.ReadEventArgs Class | SocketTools Namespace | OnRead Event (SocketTools.InternetServer)

InternetServer.ReadEventArgs Constructor

Initializes a new instance of the InternetServer.ReadEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public InternetServer.ReadEventArgs();
```

See Also

InternetServer.ReadEventArgs Class | SocketTools Namespace

InternetServer.ReadEventArgs Properties

The properties of the **InternetServer.ReadEventArgs** class are listed below. For a complete list of **InternetServer.ReadEventArgs** class members, see the InternetServer.ReadEventArgs Members topic.

Public Instance Properties

Handle	Gets a value that specifies the socket handle for the client session.

See Also

InternetServer.ReadEventArgs Class | SocketTools Namespace | OnRead Event (SocketTools.InternetServer)

InternetServer.ReadEventArgs.Handle Property

Gets a value that specifies the socket handle for the client session.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer value which specifies the client socket handle.

Remarks

The **Handle** property returns the socket handle for the client that generated the event. This handle can be used in conjunction with methods such as **Read** and **Write** to exchange data with the client.

See Also

InternetServer.ReadEventArgs Class | SocketTools Namespace

InternetServer.OnTimeout Event

Occurs when a blocking operation fails to complete before the timeout period elapses.

[Visual Basic]

```
Public Event OnTimeout As OnTimeoutEventHandler
```

[C#]

```
public event OnTimeoutEventHandler OnTimeout;
```

Event Data

The event handler receives an argument of type InternetServer.TimeoutEventArgs containing data related to this event. The following **InternetServer.TimeoutEventArgs** property provides information specific to this event.

Property	Description
Handle	Gets a value that specifies the socket handle for the client session.

Remarks

The **OnTimeout** event occurs when a blocking operation, such as sending or receiving data on the socket, fails to complete before the specified timeout period elapses. The timeout period for a blocking operation can be adjusted by setting the **Timeout** property.

User interface controls can only be accessed from the UI thread that created them, and attempting to update a control from another thread can result in the program becoming non-responsive or terminating abnormally. Because this event is generated in the context of the client thread, not the thread that created the class instance, you cannot directly modify a control from within this event handler. Instead, you must create a delegate and use the **Invoke** method to marshal invocations to the associated UI thread. For more information, refer to the documentation for the control.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.TimeoutEventArgs Class

Provides data for the OnTimeout event.

For a list of all members of this type, see InternetServer.TimeoutEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.InternetServer.TimeoutEventArgs

[Visual Basic]
```
Public Class InternetServer.TimeoutEventArgs
    Inherits EventArgs
```

[C#]
```
public class InternetServer.TimeoutEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

TimeoutEventArgs specifies the socket handle for the current client session.

The OnTimeout event occurs when a blocking operation, such as sending or receiving data on the socket, fails to complete before the specified timeout period elapses. The timeout period for a blocking operation can be adjusted by setting the **Timeout** property.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

InternetServer.TimeoutEventArgs Members | SocketTools Namespace | OnTimeout Event (SocketTools.InternetServer)

InternetServer.TimeoutEventArgs Members

InternetServer.TimeoutEventArgs overview

Public Instance Constructors

InternetServer.TimeoutEventArgs Constructor	Initializes a new instance of the InternetServer.TimeoutEventArgs class.

Public Instance Properties

Handle	Gets a value that specifies the socket handle for the client session.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetServer.TimeoutEventArgs Class | SocketTools Namespace | OnTimeout Event (SocketTools.InternetServer)

InternetServer.TimeoutEventArgs Constructor

Initializes a new instance of the InternetServer.TimeoutEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public InternetServer.TimeoutEventArgs();
```

See Also

InternetServer.TimeoutEventArgs Class | SocketTools Namespace

InternetServer.TimeoutEventArgs Properties

The properties of the **InternetServer.TimeoutEventArgs** class are listed below. For a complete list of **InternetServer.TimeoutEventArgs** class members, see the InternetServer.TimeoutEventArgs Members topic.

Public Instance Properties

Handle	Gets a value that specifies the socket handle for the client session.

See Also

InternetServer.TimeoutEventArgs Class | SocketTools Namespace | OnTimeout Event (SocketTools.InternetServer)

InternetServer.TimeoutEventArgs.Handle Property

Gets a value that specifies the socket handle for the client session.

[Visual Basic]
```
Public ReadOnly Property Handle As Integer
```

[C#]
```
public int Handle {get;}
```

Property Value

An integer value which specifies the client socket handle.

Remarks

The **Handle** property returns the socket handle for the client that generated the event. This handle can be used in conjunction with methods such as **Read** and **Write** to exchange data with the client.

See Also

InternetServer.TimeoutEventArgs Class | SocketTools Namespace

InternetServer.OnWrite Event

Occurs when data can be written to the client.

[Visual Basic]

```
Public Event OnWrite As OnWriteEventHandler
```

[C#]

```
public event OnWriteEventHandler OnWrite;
```

Event Data

The event handler receives an argument of type InternetServer.WriteEventArgs containing data related to this event. The following **InternetServer.WriteEventArgs** property provides information specific to this event.

Property	Description
Handle	Gets a value that specifies the socket handle for the client session.

Remarks

The **OnWrite** event is generated when the client can accept data from the server. The **Handle** event argument property specifies the handle to the client socket and can be used in conjunction with the **Write** or **WriteLine** methods.

This event is typically fired once when the client connection is established with the server, the session thread starts and the client socket enters a writable state. If the internal send buffer for the client socket becomes full, this event will fire again when more data can be written to the socket. It is important to note that this event is level-triggered and will not fire repeatedly if the client socket is writable. Under most circumstances this event fire only once for each client session after the initial connection has been established.

User interface controls can only be accessed from the UI thread that created them, and attempting to update a control from another thread can result in the program becoming non-responsive or terminating abnormally. Because this event is generated in the context of the client thread, not the thread that created the class instance, you cannot directly modify a control from within this event handler. Instead, you must create a delegate and use the **Invoke** method to marshal invocations to the associated UI thread. For more information, refer to the documentation for the control.

See Also

InternetServer Class | SocketTools Namespace

InternetServer.WriteEventArgs Class

Provides data for the OnWrite event.

For a list of all members of this type, see InternetServer.WriteEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.InternetServer.WriteEventArgs

[Visual Basic]

```
Public Class InternetServer.WriteEventArgs
    Inherits EventArgs
```

[C#]

```
public class InternetServer.WriteEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

WriteEventArgs specifies the socket handle for the current client session.

The **OnWrite** event is generated when the client can accept data from the server. The **Handle** event argument property specifies the handle to the client socket and can be used in conjunction with the **Write** or **WriteLine** methods.

This event is typically fired once when the client connection is established with the server, the session thread starts and the client socket enters a writable state. If the internal send buffer for the client socket becomes full, this event will fire again when more data can be written to the socket. It is important to note that this event is level-triggered and will not fire repeatedly if the client socket is writable. Under most circumstances this event fire only once for each client session after the initial connection has been established.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

InternetServer.WriteEventArgs Members | SocketTools Namespace

InternetServer.WriteEventArgs Members

InternetServer.WriteEventArgs overview

Public Instance Constructors

InternetServer.WriteEventArgs Constructor	Initializes a new instance of the InternetServer.WriteEventArgs class.

Public Instance Properties

Handle	Gets a value that specifies the socket handle for the client session.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetServer.WriteEventArgs Class | SocketTools Namespace

InternetServer.WriteEventArgs Constructor

Initializes a new instance of the InternetServer.WriteEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public InternetServer.WriteEventArgs();
```

See Also

InternetServer.WriteEventArgs Class | SocketTools Namespace

InternetServer.WriteEventArgs Properties

The properties of the **InternetServer.WriteEventArgs** class are listed below. For a complete list of **InternetServer.WriteEventArgs** class members, see the InternetServer.WriteEventArgs Members topic.

Public Instance Properties

Handle	Gets a value that specifies the socket handle for the client session.

See Also

InternetServer.WriteEventArgs Class | SocketTools Namespace

InternetServer.WriteEventArgs.Handle Property

Gets a value that specifies the socket handle for the client session.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer value which specifies the client socket handle.

Remarks

The **Handle** property returns the socket handle for the client that generated the event. This handle can be used in conjunction with methods such as **Read** and **Write** to exchange data with the client.

See Also

InternetServer.WriteEventArgs Class | SocketTools Namespace

InternetServer.OnAcceptEventHandler Delegate

Represents the method that will handle the OnAccept event.

[Visual Basic]

```
Public Delegate Sub InternetServer.OnAcceptEventHandler( _
   ByVal sender As Object, _
   ByVal e As AcceptEventArgs _
)
```

[C#]

```
public delegate void InternetServer.OnAcceptEventHandler(
      object sender,
      AcceptEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An AcceptEventArgs that contains the event data.

Remarks

When you create an **OnAcceptEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnAcceptEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

SocketTools Namespace | OnAccept Event

InternetServer.OnCancelEventHandler Delegate

Represents the method that will handle the OnCancel event.

[Visual Basic]

```
Public Delegate Sub InternetServer.OnCancelEventHandler( _
   ByVal sender As Object, _
   ByVal e As CancelEventArgs _
)
```

[C#]

```
public delegate void InternetServer.OnCancelEventHandler(
      object sender,
      CancelEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An CancelEventArgs that contains the event data.

Remarks

When you create an **OnCancelEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnCancelEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

SocketTools Namespace

InternetServer.OnConnectEventHandler Delegate

Represents the method that will handle the OnConnect event.

[Visual Basic]

```
Public Delegate Sub InternetServer.OnConnectEventHandler( _
   ByVal sender As Object, _
   ByVal e As ConnectEventArgs _
)
```

[C#]

```
public delegate void InternetServer.OnConnectEventHandler(
      object sender,
      ConnectEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ConnectEventArgs that contains the event data.

Remarks

When you create an **OnConnectEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnConnectEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

SocketTools Namespace

InternetServer.OnDisconnectEventHandler Delegate

Represents the method that will handle the OnDisconnect event.

[Visual Basic]

```
Public Delegate Sub InternetServer.OnDisconnectEventHandler( _
   ByVal sender As Object, _
   ByVal e As DisconnectEventArgs _
)
```

[C#]

```
public delegate void InternetServer.OnDisconnectEventHandler(
      object sender,
      DisconnectEventArgs e
   );
```

Parameters

sender
: The source of the event.

e
: An DisconnectEventArgs that contains the event data.

Remarks

When you create an **OnDisconnectEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnDisconnectEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

SocketTools Namespace

InternetServer.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub InternetServer.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void InternetServer.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ErrorEventArgs that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

SocketTools Namespace

InternetServer.OnReadEventHandler Delegate

Represents the method that will handle the OnRead event.

[Visual Basic]

```
Public Delegate Sub InternetServer.OnReadEventHandler( _
   ByVal sender As Object, _
   ByVal e As ReadEventArgs _
)
```

[C#]

```
public delegate void InternetServer.OnReadEventHandler(
      object sender,
      ReadEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ReadEventArgs that contains the event data.

Remarks

When you create an **OnReadEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnReadEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

SocketTools Namespace

InternetServer.OnTimeoutEventHandler Delegate

Represents the method that will handle the OnTimeout event.

[Visual Basic]

```
Public Delegate Sub InternetServer.OnTimeoutEventHandler( _
   ByVal sender As Object, _
   ByVal e As TimeoutEventArgs _
)
```

[C#]

```
public delegate void InternetServer.OnTimeoutEventHandler(
      object sender,
      TimeoutEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An TimeoutEventArgs that contains the event data.

Remarks

When you create an **OnTimeoutEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnTimeoutEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

SocketTools Namespace

InternetServer.OnWriteEventHandler Delegate

Represents the method that will handle the OnWrite event.

[Visual Basic]

```
Public Delegate Sub InternetServer.OnWriteEventHandler( _
   ByVal sender As Object, _
   ByVal e As WriteEventArgs _
)
```

[C#]

```
public delegate void InternetServer.OnWriteEventHandler(
      object sender,
      WriteEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An WriteEventArgs that contains the event data.

Remarks

When you create an **OnWriteEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnWriteEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

SocketTools Namespace

InternetServer.ErrorCode Enumeration

Specifies the error codes returned by the InternetServer class.

[Visual Basic]
```
Public Enum InternetServer.ErrorCode
```

[C#]
```
public enum InternetServer.ErrorCode
```

Remarks

The InternetServer class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.
errorProxyAuthenticationRequired	Proxy authentication required.

errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.
errorUnsupportedMediaType	Request specified an unsupported media type.

errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.
errorNoMoreMailboxes	No more mailboxes exist on this server.

errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.
errorInvalidThreadHandle	The specified thread handle is no longer valid.

errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been defined.

errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or unrecognized.

errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

InternetServer.SecurityProtocols Enumeration

Specifies the security protocols that the InternetServer class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum InternetServer.SecurityProtocols
```

[C#]

```
[Flags]
public enum InternetServer.SecurityProtocols
```

Remarks

The InternetServer class uses the **SecurityProtocols** enumeration to specify one or more security protocols to be used when establishing a connection with a remote host. Multiple protocols may be specified if necessary and the actual protocol used will be negotiated with the remote host. It is recommended that most applications use **protocolDefault** when starting a secure server.

Members

Member Name	Description	Value
protocolNone	No security protocol will be used, a secure connection will not be established.	0
protocolSSL2	The SSL 2.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections.	1
protocolSSL3	The SSL 3.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections. In most cases, this protocol is only selected if TLS is not supported by the server.	2
protocolTLS10	The TLS 1.0 protocol has been selected. This version of the protocol is commonly used by older servers and is the only version of TLS supported on Windows platforms prior to Windows 7 SP1 and Windows Server 2008 R2.	4
protocolTLS11	The TLS 1.1 protocol has been selected. This version of TLS is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of the	8

	operating system.	
protocolTLS12	The TLS 1.2 protocol has been selected. This is the default version of the protocol and is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of Windows. It is recommended that you use this version of TLS.	16
protocolSSL	Any version of the Secure Sockets Layer (SSL) protocol should be used. The actual protocol version used will be negotiated with the remote host.	3
protocolTLS	Any version of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with with preference for TLS 1.2.	28
protocolTLS1	Version 1.0, 1.1 or 1.2 of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with preference for TLS 1.2.	28
protocolDefault	The default selection of security protocols will be used when establishing a connection. The TLS 1.2, 1.1 and 1.0 protocols will be negotiated with the host, in that order of preference. This option will always request the latest version of the preferred security protocols and is the recommended value.	16
protocolUnknown	An unknown or unsupported security protocol has been specified. This value indicates an error condition.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

SocketTools Namespace

InternetServer.ServerOptions Enumeration

Specifies the options that the InternetServer class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum InternetServer.ServerOptions
```

[C#]

```
[Flags]
public enum InternetServer.ServerOptions
```

Remarks

The InternetServer class uses the **ServerOptions** enumeration to specify one or more options to be used when establishing a connection with a remote host. Multiple options may be specified if necessary.

Members

Member Name	Description	Value
optionNone	No option specified.	0
optionDontRoute	This option specifies default routing should not be used. This option should not be specified unless absolutely necessary.	2
optionKeepAlive	This option specifies that packets are to be sent to the remote system when no data is being exchanged to keep the connection active. This option is only valid for stream sockets.	4
optionReuseAddress	This option specifies the local address can be reused. This option is commonly used by server applications.	8
optionNoDelay	This option disables the Nagle algorithm, which buffers unacknowledged data and insures that a full-size packet can be sent to the remote host.	16
optionSecure	This option specifies that a secure, encrypted connection will be established with the remote host.	4096
optionSecureFallback	This option specifies the server should permit the use of less secure cipher suites for compatibility with legacy clients. If this option is specified, the server will permit connections using TLS 1.0 and cipher suites that use RC4, MD5 and SHA1.	32768

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

SocketTools Namespace

InternetServer.ServerPriority Enumeration

Specifies the priorities that the InternetServer class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum InternetServer.ServerPriority
```

[C#]

```
[Flags]
public enum InternetServer.ServerPriority
```

Members

Member Name	Description	Value
priorityBackground	This priority significantly reduces the memory, processor and network resource utilization for the server. It is typically used with lightweight services running in the background that are designed for few client connections. Each client thread will be assigned a lower scheduling priority and will be frequently forced to yield execution to other threads.	0
priorityLow	This priority lowers the overall resource utilization for the client session and meters the processor utilization for the client session. Each client thread will be assigned a lower scheduling priority and will occasionally be forced to yield execution to other threads.	1
priorityNormal	The default priority which balances resource and processor utilization. It is recommended that most applications use this priority.	2
priorityHigh	This priority increases the overall resource utilization for each client session and their threads will be given higher scheduling priority. It is not recommended that this priority be used on a system with a single processor.	3
priorityCritical	This priority can significantly increase processor, memory and network utilization. Each client thread will be given higher scheduling priority and will be more responsive to network events. It is not recommended that this priority be used on a system with a single	4

	processor.	
priorityInvalid	An invalid transfer priority which indicates an error condition.	-1
priorityDefault	The default server priority. This is the same as specifying **priorityNormal**.	2
priorityLowest	The lowest valid server priority. This is the same as specifying **priorityBackground**.	0
priorityHighest	The highest valid server priority. This is the same as specifying **priorityCritical**.	4

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

SocketTools Namespace

InternetServer.ServerStatus Enumeration

Specifies the status values that may be returned by the InternetServer class.

[Visual Basic]
```
Public Enum InternetServer.ServerStatus
```

[C#]
```
public enum InternetServer.ServerStatus
```

Remarks

The InternetServer class uses the **ServerStatus** enumeration to identify the current status of the server.

Members

Member Name	Description
serverInactive	The server is currently inactive. This status is returned when no server has been started, or after a server has been stopped.
serverStarted	The server has initialized and is preparing to listen for client connections. This status is returned after the server thread has been started, but before the listening socket has been created.
serverListening	The server is actively listening for incoming client connections. This status is returned after the server thread has been started and the listening socket has been created.
serverSuspended	The server has been suspended and is no longer accepting client connections. Any incoming client connection is queued, and will be accepted when the server resumes normal operation.
serverShutdown	The server has been stopped and is in the process of terminating all active client sessions.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

SocketTools Namespace

InternetServer.TraceOptions Enumeration

Specifies the logging options that the InternetServer class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum InternetServer.TraceOptions
```

```
[C#]
[Flags]
public enum InternetServer.TraceOptions
```

Remarks

The InternetServer class uses the **TraceOptions** enumeration to specify what kind of debugging information is written to the trace logfile. These options are only meaningful when trace logging is enabled by setting the **Trace** property to **true**.

Members

Member Name	Description	Value
traceDefault	The default trace logging option. This is the same as specifying the **traceInfo** option.	0
traceInfo	All network function calls are written to the trace file. This is the default value.	0
traceError	Only those network function calls which fail are recorded in the trace file.	1
traceWarning	Only those network function calls which fail, or return values which indicate a warning, are recorded in the trace file.	2
traceHexDump	All network function calls are written to the trace file, plus all the data that is sent or received is displayed, in both ASCII and hexadecimal format.	4
traceProcess	All function calls in the current process are logged, rather than only those functions in the current thread. This option is useful for multithreaded applications that are using worker threads.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

SocketTools Namespace

InternetServer.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see InternetServer.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.InternetServer.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class InternetServer.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class InternetServer.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the InternetServer class.

Example

```
<Assembly: SocketTools.InternetServer.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.InternetServer.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

InternetServer.RuntimeLicenseAttribute Members | SocketTools Namespace

InternetServer.RuntimeLicenseAttribute Members

InternetServer.RuntimeLicenseAttribute overview

Public Instance Constructors

InternetServer.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetServer.RuntimeLicenseAttribute Class | SocketTools Namespace

InternetServer.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public InternetServer.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the InternetServer class.

See Also

InternetServer.RuntimeLicenseAttribute Class | SocketTools Namespace

InternetServer.RuntimeLicenseAttribute Properties

The properties of the **InternetServer.RuntimeLicenseAttribute** class are listed below. For a complete list of **InternetServer.RuntimeLicenseAttribute** class members, see the InternetServer.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

InternetServer.RuntimeLicenseAttribute Class | SocketTools Namespace

InternetServer.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

InternetServer.RuntimeLicenseAttribute Class | SocketTools Namespace

InternetServerException Class

The exception that is thrown when a socket error occurs.

For a list of all members of this type, see InternetServerException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.InternetServerException

[Visual Basic]
```
Public Class InternetServerException
    Inherits ApplicationException
```

[C#]
```
public class InternetServerException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A InternetServerException is thrown by the InternetServer class when an error occurs.

The default constructor for the InternetServerException class sets the **ErrorCode** property to the last socket error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.InternetServer (in SocketTools.InternetServer.dll)

See Also

InternetServerException Members | SocketTools Namespace

InternetServerException Members

InternetServerException overview

Public Instance Constructors

InternetServerException	Overloaded. Initializes a new instance of the InternetServerException class.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
Handle	Gets a value which specifies the socket handle that generated the error.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetServerException Class | SocketTools Namespace

InternetServerException Constructor

Initializes a new instance of the InternetServerException class with the last network error code.

Overload List

Initializes a new instance of the InternetServerException class with the last network error code.

public InternetServerException();

Initializes a new instance of the InternetServerException class with the last network error code for the specified socket.

public InternetServerException(int);

Initializes a new instance of the InternetServerException class with a specified error number.

public InternetServerException(int,int);

Initializes a new instance of the InternetServerException class with a specified error message.

public InternetServerException(int,string);

Initializes a new instance of the InternetServerException class with a specified error message and a reference to the inner exception that is the cause of this exception.

public InternetServerException(int,string,Exception);

See Also

InternetServerException Class | SocketTools Namespace

InternetServerException Constructor ()

Initializes a new instance of the InternetServerException class with the last network error code.

[Visual Basic]

```
Overloads Public Sub New()
```

[C#]

```
public InternetServerException();
```

Remarks

The ctor constructor sets the **ErrorCode** property to the last socket error that occurred. For more information about the errors that may occur, refer to the InternetServer.ErrorCode enumeration.

See Also

InternetServerException Class | SocketTools Namespace | InternetServerException Constructor Overload List

InternetServerException Constructor (Int32)

Initializes a new instance of the InternetServerException class with the last network error code for the specified socket.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal handle As Integer _
)
```

[C#]

```
public InternetServerException(
   int handle
);
```

Parameters

handle

An integer value which specifies the handle to the socket which generated the error.

Remarks

The ctor constructor sets the **ErrorCode** property to the last socket error that occurred. For more information about the errors that may occur, refer to the InternetServer.ErrorCode enumeration.

See Also

InternetServerException Class | SocketTools Namespace | InternetServerException Constructor Overload List

InternetServerException Constructor (Int32, String)

Initializes a new instance of the InternetServerException class with a specified error message.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal handle As Integer, _
   ByVal message As String _
)
```

[C#]

```
public InternetServerException(
   int handle,
   string message
);
```

Parameters

handle

An integer value which specifies the handle to the socket which generated the error.

message

The error message that explains the reason for the exception.

Remarks

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

InternetServerException Class | SocketTools Namespace | InternetServerException Constructor Overload List

InternetServerException Constructor (Int32, String, Exception)

Initializes a new instance of the InternetServerException class with a specified error message and a reference to the inner exception that is the cause of this exception.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal handle As Integer, _
   ByVal message As String, _
   ByVal innerException As Exception _
)
```

[C#]

```
public InternetServerException(
   int handle,
   string message,
   Exception innerException
);
```

Parameters

handle

An integer value which specifies the handle to the socket which generated the error.

message

The error message that explains the reason for the exception.

innerException

The exception that is the cause of the current exception. If the *innerException* parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

InternetServerException Class | SocketTools Namespace | InternetServerException Constructor Overload List

InternetServerException Properties

The properties of the **InternetServerException** class are listed below. For a complete list of **InternetServerException** class members, see the InternetServerException Members topic.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
Handle	Gets a value which specifies the socket handle that generated the error.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

InternetServerException Class | SocketTools Namespace

InternetServerException.ErrorCode Property

Gets a value which specifies the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property ErrorCode As ErrorCode
```

[C#]

```
public InternetServer.ErrorCode ErrorCode {get;}
```

Property Value

Returns a InternetServer.ErrorCode enumeration value which specifies the error code.

Remarks

The **ErrorCode** property returns the error code that specifies the cause of the exception.

The default constructor for the InternetServerException class sets the error code to the last network error that occurred. For more information about the errors that may occur, refer to the InternetServer.ErrorCode enumeration.

See Also

InternetServerException Class | SocketTools Namespace

InternetServerException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]

```
Overrides Public ReadOnly Property Message As String
```

[C#]

```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

InternetServerException Class | SocketTools Namespace

InternetServerException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception. This value may be the same as values returned by the Windows Sockets API. For more information about socket error codes, see the Windows Socket Version 2 API error code documentation in MSDN.

See Also

InternetServerException Class | SocketTools Namespace

InternetServerException Methods

The methods of the **InternetServerException** class are listed below. For a complete list of **InternetServerException** class members, see the InternetServerException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

InternetServerException Class | SocketTools Namespace

InternetServerException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]
```
Overrides Public Function ToString() As String
```

[C#]
```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

InternetServerException Class | SocketTools Namespace

MailMessage Class

Implements the Multipurpose Internet Mail Extensions standard.

For a list of all members of this type, see MailMessage Members.

System.Object
  SocketTools.MailMessage

[Visual Basic]
```
Public Class MailMessage
    Implements IDisposable
```

[C#]
```
public class MailMessage : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The MailMessage class provides an interface for composing and processing email messages and newsgroup articles which are structured according to the Multipurpose Internet Mail Extensions (MIME) standard. Using this class, an application can easily create complex messages which include multiple alternative content types, such as plain text and styled HTML text, file attachments and customized headers.

It is not required that the developer understand the complex MIME standard; a single method can be used to create multipart message, complete with a styled HTML text body and support for international character sets. The MailMessage class can be easily integrated with the other mail related protocol classes, making it extremely easy to create and process MIME formatted messages.

Requirements

Namespace: SocketTools

Assembly: SocketTools.MailMessage (in SocketTools.MailMessage.dll)

See Also

MailMessage Members | SocketTools Namespace

MailMessage Members

MailMessage overview

Public Instance Constructors

MailMessage Constructor	Initializes a new instance of the MailMessage class.

Public Instance Fields

Recipient	Gets the recipients specified in the current message.

Public Instance Properties

AllHeaders	Gets a value which returns a list of all message recipients.
AllRecipients	Gets a value which returns a list of all message recipients.
Attachment	Gets and sets the name of the current file attachment.
Bcc	Gets and sets the blind carbon-copy header field value.
Boundary	Gets the boundary string used to separate parts in a multipart message.
Cc	Gets and sets the carbon-copy header field value.
ContentId	Gets the content identifier for the current message part.
ContentLength	Gets the size of the data stored in the current message part.
ContentType	Gets and sets the content type of the selected message part.
Date	Gets and sets the date for the current message.
Encoding	Gets and sets the content encoding method used for the current message part.
From	Gets and sets the address of the user who sent the message.
Handle	Gets a value that specifies the client handle allocated for the current message.
HeaderField	Gets and sets the current header field name.
HeaderValue	Gets and sets the value of the current header field.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
LastError	Gets and sets a value which specifies the last error that has occurred.

LastErrorString	Gets a value which describes the last error that has occurred.
Localize	Gets and sets a value which specifies if date and time values should be localized.
Mailer	Gets and sets the name of the mailer application.
Message	Gets and sets the current message headers and body.
MessageID	Gets the current message identifier.
MimeVersion	Gets and sets the MIME version number for the current message.
Options	Gets and sets a value which specifies one or more message export options.
Organization	Gets and sets the name of the organization that originated the message.
Part	Gets and sets the current message part.
PartCount	Gets the number of parts in the current message.
Priority	Gets and sets the current message priority.
Recipients	Gets the number of recipients specified in the current message.
ReplyTo	Gets and sets the address of the user who should receive replies to this message.
Sender	Gets and sets the address of the user who originated the message.
StoreCount	Gets the number of messages in the current storage file, not including deleted messages.
StoreFile	Gets and sets the name of the file used to store messages.
StoreIndex	Gets and sets the current message index for the current storage file.
StoreSize	Gets the total number of messages in the current storage file, including deleted messages.
Subject	Gets and sets the subject of the current message.
Text	Gets and sets the text body of the current message part.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
TimeZone	Gets and sets the current timezone offset in seconds.
To	Gets and sets the address of the message recipient.

Version	Gets a value which returns the current version of the MailMessage class library.

Public Instance Methods

AppendMessage	Append text to the body of the current message part.
AttachData	Overloaded. Attach the contents of a byte array to the current message.
AttachFile	Overloaded. Attach the specified file to the current message.
AttachImage	Overloaded. Attach an inline image to the current message.
AttachThread	Obsolete. Attach an instance of the class to the current thread.
ClearMessage	Clear the header and body of the current message.
CloseStore	Close the current message storage file.
ComposeMessage	Overloaded. Compose a new mail message.
CreatePart	Overloaded. Create a new message part in a multipart message.
DecodeText	Overloaded. Decode a string which was previously encoded using base64 or quoted-printable encoding.
DeleteHeader	Overloaded. Delete a header field from the specified message part.
DeleteMessage	Overloaded. Remove the specified message from the current message store.
DeletePart	Delete the specified message part from the current message.
Dispose	Overloaded. Releases all resources used by MailMessage.
EncodeText	Overloaded. Encodes a string using base64 or quoted-printable encoding.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
ExportMessage	Overloaded. Export the current message to a file on the local system.
ExtractAllFiles	Overloaded. Extract all file attachments from the current message.
ExtractFile	Overloaded. Extract the contents of a file attachment and store it on the local system.
FindAttachment	Search for a specific file attachment in the current message.

FindMessage	Overloaded. Search for a message in the current message store.
GetFirstHeader	Return the first header in the current message part.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetHeader	Overloaded. Return the value of a header field in the specified message part.
GetNextHeader	Return the next header in the current message part.
GetType (inherited from Object)	Gets the Type of the current instance.
ImportMessage	Replace the current message with the contents of a file.
Initialize	Overloaded. Initialize an instance of the MailMessage class.
OpenStore	Overloaded. Open the specified message storage file.
ParseAddress	Overloaded. Parse an Internet email address.
ParseMessage	Parse the specified string, adding the contents to the current message.
PurgeStore	Purge all deleted messages from the current message store.
ReadStore	Overloaded. Retrieve a message from the message store, replacing the current message.
ReplaceMessage	Overloaded. Replace the specified message in the current message store.
Reset	Reset the internal state of the object, resetting all properties to their default values.
SetHeader	Overloaded. Set the value for a header in the specified message part.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
WriteStore	Store the current message in the message store.

Public Instance Events

OnError	Occurs when an client operation fails.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the MailMessage class and optionally

	releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the current message.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

MailMessage Class | SocketTools Namespace

MailMessage Constructor

Initializes a new instance of the MailMessage class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public MailMessage();
```

See Also

MailMessage Class | SocketTools Namespace

MailMessage Properties

The properties of the **MailMessage** class are listed below. For a complete list of **MailMessage** class members, see the MailMessage Members topic.

Public Instance Properties

AllHeaders	Gets a value which returns a list of all message recipients.
AllRecipients	Gets a value which returns a list of all message recipients.
Attachment	Gets and sets the name of the current file attachment.
Bcc	Gets and sets the blind carbon-copy header field value.
Boundary	Gets the boundary string used to separate parts in a multipart message.
Cc	Gets and sets the carbon-copy header field value.
ContentId	Gets the content identifier for the current message part.
ContentLength	Gets the size of the data stored in the current message part.
ContentType	Gets and sets the content type of the selected message part.
Date	Gets and sets the date for the current message.
Encoding	Gets and sets the content encoding method used for the current message part.
From	Gets and sets the address of the user who sent the message.
Handle	Gets a value that specifies the client handle allocated for the current message.
HeaderField	Gets and sets the current header field name.
HeaderValue	Gets and sets the value of the current header field.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
Localize	Gets and sets a value which specifies if date and time values should be localized.
Mailer	Gets and sets the name of the mailer application.

Message	Gets and sets the current message headers and body.
MessageID	Gets the current message identifier.
MimeVersion	Gets and sets the MIME version number for the current message.
Options	Gets and sets a value which specifies one or more message export options.
Organization	Gets and sets the name of the organization that originated the message.
Part	Gets and sets the current message part.
PartCount	Gets the number of parts in the current message.
Priority	Gets and sets the current message priority.
Recipients	Gets the number of recipients specified in the current message.
ReplyTo	Gets and sets the address of the user who should receive replies to this message.
Sender	Gets and sets the address of the user who originated the message.
StoreCount	Gets the number of messages in the current storage file, not including deleted messages.
StoreFile	Gets and sets the name of the file used to store messages.
StoreIndex	Gets and sets the current message index for the current storage file.
StoreSize	Gets the total number of messages in the current storage file, including deleted messages.
Subject	Gets and sets the subject of the current message.
Text	Gets and sets the text body of the current message part.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
TimeZone	Gets and sets the current timezone offset in seconds.
To	Gets and sets the address of the message recipient.
Version	Gets a value which returns the current version of the MailMessage class library.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.AllHeaders Property

Gets a value which returns a list of all message recipients.

[Visual Basic]

```
Public ReadOnly Property AllHeaders As String
```

[C#]

```
public string AllHeaders {get;}
```

Property Value

A string which contains the complete RFC822 headers for the message.

Remarks

The **AllHeaders** property will return all of the RFC 822 header values in a string. This includes the message headers that are most commonly referred to, such as the To, From and Subject headers. Each header and its value are separated by a colon, and terminated with a carriage return and linefeed (CRLF) pair.

The headers and their values returned by this property will not be identical to the header block in the original message. If a header value is split across multiple lines, the text returned by this property will be folded, with the complete header value on a single line of text and removing any extraneous whitespace. If the header value has been encoded by the mail client, this property will return the decoded value, not the original encoded value.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.AllRecipients Property

Gets a value which returns a list of all message recipients.

[Visual Basic]

```
Public ReadOnly Property AllRecipients As String
```

[C#]

```
public string AllRecipients {get;}
```

Property Value

A string which contains a comma-separated list of all message recipients.

Remarks

The **AllRecipients** property returns a string value that contains a comma-separated list of all message recipients. To individually enumerate through each of the recipient addresses, you can use the **Recipient** property array and **Recipients** property.

Note that this property value will include those addresses specified by the **Bcc** property, even though they are not included in the message header.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Attachment Property

Gets and sets the name of the current file attachment.

[Visual Basic]

```
Public ReadOnly Property Attachment As String
```

[C#]

```
public string Attachment {get;}
```

Property Value

A string which specifies the name of an attached file.

Remarks

The **Attachment** property specifies the name of the file attachment in a multipart message. When a new part is selected that contains an attached file, the **Attachment** property is updated to reflect the attached file's name.

This property is used by the attach and extract actions to specify the local file name that will be used. Changing its value does not change the attached file name in the multipart message itself.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Bcc Property

Gets and sets the blind carbon-copy header field value.

[Visual Basic]

```
Public Property Bcc As String
```

[C#]

```
public string Bcc {get; set;}
```

Property Value

A string which specifies one or more blind carbon-copy recipients.

Remarks

The **Bcc** property returns the list of addresses that are to receive blind carbon copies of the message. Setting the property creates or modifies the Bcc header field. Multiple addresses can be specified by separating them with commas.

A blind carbon copy is when a copy of a message is delivered to a recipient, but that recipient is not listed in the message headers. Because the other recipients of that same message will not see the address in the headers, they will not know it was delivered to that person. As a result, the Bcc header field is not normally exported when the **ExportMessage** method is called, or when the contents of the message are referenced using the **Message** property. To include the Bcc header in the message, use the **exportAllHeaders** option. Of course, if this option is specified, the addresses in the Bcc list will no longer be blind to the other recipients.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Boundary Property

Gets the boundary string used to separate parts in a multipart message.

[Visual Basic]

```
Public ReadOnly Property Boundary As String
```

[C#]

```
public string Boundary {get;}
```

Property Value

A string which specifies the current boundary in a multipart message.

Remarks

The **Boundary** property returns the current boundary string being used in a multipart message. When the class is used to create a multipart message, a unique boundary string is created and the **Boundary** property is updated to reflect it's value.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Cc Property

Gets and sets the carbon-copy header field value.

[Visual Basic]
```
Public Property Cc As String
```

[C#]
```
public string Cc {get; set;}
```

Property Value

A string which specifies one or more carbon-copy recipients.

Remarks

The **Cc** property returns the list of addresses that were delivered carbon copies of the message. Setting the property creates or modifies the Cc header field. Multiple addresses can be specified by separating them with commas.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.ContentId Property

Gets the content identifier for the current message part.

[Visual Basic]

```
Public ReadOnly Property ContentId As String
```

[C#]

```
public string ContentId {get;}
```

Property Value

A string which specifies the content identifier.

Remarks

The **ContentId** property returns the unique content identifier string for the current message part. This multipart header field is not commonly used, and if undefined, will return an empty string.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.ContentLength Property

Gets the size of the data stored in the current message part.

[Visual Basic]

```
Public ReadOnly Property ContentLength As Integer
```

[C#]

```
public int ContentLength {get;}
```

Property Value

An integer which specifies the size of the current message part in bytes.

Remarks

The **ContentLength** property returns the size of the data stored in the selected message part. This property is read-only, and is updated when the current message part changes.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.ContentType Property

Gets and sets the content type of the selected message part.

[Visual Basic]
```
Public Property ContentType As String
```

[C#]
```
public string ContentType {get; set;}
```

Property Value

A string which specifies the content type.

Remarks

The **ContentType** property returns the MIME type for the currently selected message part. The type string consists of a primary type and secondary sub-type separated by a slash, followed by one or more optional parameters delimited by semi-colons. For example, this is a common content type for text messages:

```
    text/plain; charset=utf-8
```

The **text** designation indicates that this message part contains readable text, and the **plain** sub-type indicates that the text does not contain any special encoding. The optional parameter which follows the content type provides additional information about the content. In this example, it specifies which character set should be used to display the text. The two common character sets used are UTF-8 and US-ASCII.

There are seven predefined, standard content types, each with their own sub-types. The following table lists these types, along with some common sub-types that are found in messages:

Type	Description
text	Indicates that the message part contains text. This is the most common type found in mail messages; if no content type is explicitly defined, then it is assumed to be plain text. Examples are text/plain, text/richtext and text/html.
image	Indicates that the message part contains a graphics image. Examples are image/gif and image/jpeg.
audio	Indicates that the message part contains audio data; the basic sub-type is 8-bit PCM encoded audio (commonly found with the .au filename extension). Examples are audio/basic, audio/aiff and audio/wav.
video	Indicates that the message part contains a video clip in the specified format. Examples are video/mpeg and video/avi.
application	Indicates that the message part contains application specific data, typically used with the octet-stream sub-type to indicate binary file attachments for executable programs, compressed

	file archives, etc. Examples are application/octet-stream and application/postscript.
message	Indicates that the message part contains a complete RFC 822 compliant message, complete with headers. An example is message/rfc822.
multipart	Indicates that this is part of a mixed message (a message that contains multiple parts of different content types). Examples are multipart/alternative and multipart/mixed.

The three most common content types that are used in applications are text/plain for the mail message body, application/octet-stream for binary file attachments and multipart/mixed for messages that contain both text and attached files. For more information about the different content types, refer to the Multipurpose Internet Mail Extensions (MIME) standards document RFC 1521.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Date Property

Gets and sets the date for the current message.

[Visual Basic]

```
Public Property Date As String
```

[C#]

```
public string Date {get; set;}
```

Property Value

A string which specifies the date in RFC 822 format.

Remarks

The **Date** property returns the value of the date field in the current message header. Setting this property causes the date field to be updated with the specified value. When setting the date, any one of the following formats may be used:

Format	Example
mm/dd/yy[yy] hh:mm[:ss]	03/01/2006 12:00:00
yy[yy]/mm/dd hh:mm[:ss]	2006/03/01 12:00:00
dd mmm yy[yy] hh:mm[:ss]	01 Mar 2006 12:00:00
mmm dd yy[yy] hh:mm[:ss]	Mar 01 2006 12:00:00

Any extraneous information that may be included in the date string, such as the day of the week, is ignored. In addition to the date and time, the string may also include a time zone specification at the end. If no time zone is specified, the current time zone is used.

When specifying a time zone, the value should either be prefixed by a plus sign (+) to indicate that the time zone is to the east of GMT, or a minus sign (-) to indicates that it's to the west. Four digits follow, with the first two indicating the number of hours east or west of GMT, and the last two digits indicating the number of minutes. Therefore, a value of -0800 means that the time zone is eight hours to the west of GMT, or in other words, the Pacific time zone.

Regardless of the format of the string assigned to the property, it always returns the date in the same format (which conforms to the RFC 822 specification). Using the above examples, the date would be returned as "Wed, 01 Mar 2006 12:00:00 -0800" if you are located in the Pacific timezone.

The **Localize** property affects how dates are processed by the control. If enabled, dates are automatically adjusted for the local time zone. By default, localization is disabled.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Encoding Property

Gets and sets the content encoding method used for the current message part.

[Visual Basic]
```
Public Property Encoding As String
```

[C#]
```
public string Encoding {get; set;}
```

Property Value

A string which specifies the encoding type.

Remarks

The **Encoding** property returns the method used for encoding the current message part. Setting this property causes the Content-Transfer-Encoding header value to be updated. The following values are commonly used:

Type	Description
7bit	Printable ASCII text characters that only use 7-bit characters This may be used for backwards compatibility with older mail servers that do not support 8-bit message text.
8bit	Printable ANSI characters, including those characters with the high-bit set. This includes UTF-8 and the ISO Latin-1 character set.
binary	All characters; binary transfer encoding is rarely used.
quoted-printable	Printable ASCII characters, with non-printable or extended characters represented using their hexadecimal equivalents.
base64	The transfer encoding type commonly used to convert binary data into 7-bit ASCII characters so that it may be transported safely through the mail system.
x-uuencode	A transfer encoding type similar in function to the base64 encoding method.

Note that setting this property only updates the Content-Transfer-Encoding header value. To control the actual encoding method used for attachments, specify the encoding method when calling the **AttachFile** method.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.From Property

Gets and sets the address of the user who sent the message.

[Visual Basic]

```
Public Property From As String
```

[C#]

```
public string From {get; set;}
```

Property Value

A string which specifies the sender of the message.

Remarks

The **From** property returns the address of the user who sent the message. Setting the property causes the From header field to be updated with the new value.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Handle Property

Gets a value that specifies the client handle allocated for the current message.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer which represents a client handle. If there is no current message, a value of -1 is returned.

Remarks

The **Handle** property specifies the numeric descriptor of the current message.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.HeaderField Property

Gets and sets the current header field name.

[Visual Basic]
```
Public Property HeaderField As String
```

[C#]
```
public string HeaderField {get; set;}
```

Property Value

A string which specifies the current header field.

Remarks

The **HeaderField** property returns the name of the current header field. Setting this property causes the control to determine if that header field exists, and if it does, to update the **HeaderValue** property with it's value. This property can be used to obtain the value of any header in the current message part, and in conjunction with the **HeaderValue** property, can be used to create new headers.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.HeaderValue Property

Gets and sets the value of the current header field.

[Visual Basic]

```
Public Property HeaderValue As String
```

[C#]

```
public string HeaderValue {get; set;}
```

Property Value

A string which specifies the value of the current header field.

Remarks

The **HeaderValue** property returns the value of the header specified by the **HeaderField** property. Setting this property updates the specified header value. If the **HeaderField** property refers to a header field that does not exist, then it is created in the current message part.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.IsInitialized Property

Gets a value which indicates if the current instance of the class has been initialized successfully.

[Visual Basic]

```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]

```
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the class instance has been initialized; otherwise returns **false**.

Remarks

The **IsInitialized** property is used to determine if the current instance of the class has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the class may not be able to initialize itself.

The most common reasons that a class instance may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required libraries or not being able to access the system registry.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public MailMessage.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]
```
Public ReadOnly Property LastErrorString As String
```

[C#]
```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Localize Property

Gets and sets a value which specifies if date and time values should be localized.

[Visual Basic]

```
Public Property Localize As Boolean
```

[C#]

```
public bool Localize {get; set;}
```

Property Value

A boolean value which specifies if date and time values should be localized.

Remarks

The **Localize** property is used to enable or disable localization features of the class. Currently this only affects the way in which dates are processed by the class. If set to **true**, the control will adjust for the local time zone when setting and reading the **Date** property. The default value for this property is **false**.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Mailer Property

Gets and sets the name of the mailer application.

[Visual Basic]

```
Public Property Mailer As String
```

[C#]

```
public string Mailer {get; set;}
```

Property Value

A string which specifies the name of the mailer application.

Remarks

The **Mailer** property returns the value of the X-Mailer field in the current message header. Setting this property causes the field to be updated with the specified value. This is typically used to identify the program which generated the message.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Message Property

Gets and sets the current message headers and body.

[Visual Basic]

```
Public Property Message As String
```

[C#]

```
public string Message {get; set;}
```

Property Value

A string which contains the complete message.

Remarks

The **Message** property returns the current message, including the headers and all message parts, as a string. Setting this property will cause the current message to be cleared and replaced by the new value. The contents must follow the standard specifications for a message. If the property is set to an empty string, the current message is cleared.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.MessageID Property

Gets the current message identifier.

[Visual Basic]

```
Public Property MessageID As String
```

[C#]

```
public string MessageID {get; set;}
```

Property Value

A string which specifies the message identifier.

Remarks

The **MessageID** property returns a unique string that can be used to identify the message.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.MimeVersion Property

Gets and sets the MIME version number for the current message.

[Visual Basic]

```
Public Property MimeVersion As String
```

[C#]

```
public string MimeVersion {get; set;}
```

Property Value

A string that specifies the version number.

Remarks

The **MimeVersion** property returns the version number for the current message. Setting this property causes the MIME-Version header value to be changed to the specified value. An empty string causes the MIME version number to be set to the default value of "1.0". It is recommended that you do not change the value of this property.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Options Property

Gets and sets a value which specifies one or more message export options.

[Visual Basic]

```
Public Property Options As MimeExportOptions
```

[C#]

```
public MailMessage.MimeExportOptions Options {get; set;}
```

Property Value

Returns one or more MimeExportOptions enumeration flags which specify the options for the exporting the contents of a message. The default value for this property is **exportDefault**.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Organization Property

Gets and sets the name of the organization that originated the message.

[Visual Basic]
```
Public Property Organization As String
```

[C#]
```
public string Organization {get; set;}
```

Property Value

A string which specifies the organization name.

Remarks

The **Organization** property returns the name of the organization that originated the current message. Setting this property updates the specified header value. Note that many mail clients do not generate an Organization header field, in which case the property value will be an empty string.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Part Property

Gets and sets the current message part.

[Visual Basic]

```
Public Property Part As Integer
```

[C#]

```
public int Part {get; set;}
```

Property Value

An integer which specifies the current message part.

Remarks

The **Part** property returns the current message part index. All messages have at least one part, which consists of one or more header fields, followed by the body of the message. The default part, part 0, refers to the main message header and body. If the message contains multiple parts (as with a message that contains one or more attached files), the **Part** property can be set to refer to that specific part of the message.

For example, messages with file attachments typically consist of a message part which describes the contents of the attachment, followed by the attachment itself. For a message with one attached file, there would be a total of three parts. Part 0 would refer to the main message part, which contains the headers such as From, To, Subject, Date and so on. For multipart messages, part 0 typically does not have a message body, since any text is usually created as a separate part (for those messages that do not contain multiple parts, the part 0 body contains the text message). Part 1 would contain the text describing the attachment, and part 2 would contain the attachment itself. If the attached file is binary, then the transfer encoding type would usually be base64.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.PartCount Property

Gets the number of parts in the current message.

[Visual Basic]

```
Public ReadOnly Property PartCount As Integer
```

[C#]

```
public int PartCount {get;}
```

Property Value

An integer value which specifies the number of message parts.

Remarks

The **PartCount** property returns the number of parts in the current message. All messages have at least one part, referenced as part 0. Multipart messages will consist of additional parts which may be accessed by setting the **Part** property.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Priority Property

Gets and sets the current message priority.

[Visual Basic]

```
Public Property Priority As String
```

[C#]

```
public string Priority {get; set;}
```

Property Value

A string which specifies the message priority.

Remarks

The **Priority** property returns the current priority for the message. Setting this property value causes the X-Priority header to be updated with the specified value.

There is no standard for specifying message priority. The convention is to use a number from 1-5, with 1 indicating the highest priority, 3 as normal priority and 5 as the lowest priority. Some mailers follow the number with a space and then text that describes the priority level.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Recipient Field

Gets the recipients specified in the current message.

[Visual Basic]

```
Public ReadOnly Recipient As RecipientArray
```

[C#]

```
public readonly RecipientArray Recipient;
```

Remarks

The **Recipient** array is used to enumerate the recipient addresses that have been specified in the current message. This includes all of the addresses listed in the To, Cc and Bcc header fields. Only the address itself will be returned, not any comments or extraneous text such as the full name of the recipient. This array is zero based, meaning that the first index value is zero. The total number of recipients specified in the message can be determined by checking the value of the **Recipients** property.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Recipients Property

Gets the number of recipients specified in the current message.

[Visual Basic]

```
Public ReadOnly Property Recipients As Integer
```

[C#]

```
public int Recipients {get;}
```

Property Value

An integer which specifies the number of recipients.

Remarks

The **Recipients** property returns the number of recipient addresses that have been specified in the current message. This includes all of the addresses listed in the To, Cc and Bcc header fields. This property can be used in conjunction with the **Recipient** array to enumerate all of the recipient addresses in the message.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.ReplyTo Property

Gets and sets the address of the user who should receive replies to this message.

[Visual Basic]

```
Public Property ReplyTo As String
```

[C#]

```
public string ReplyTo {get; set;}
```

Property Value

A string that specifies an email address.

Remarks

The **ReplyTo** property returns the address of the user who should receive replies to the current message. Setting this property updates the Reply-To header with the specified value.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Sender Property

Gets and sets the address of the user who originated the message.

[Visual Basic]
```
Public Property Sender As String
```

[C#]
```
public string Sender {get; set;}
```

Property Value

A string which specifies the sender's email address.

Remarks

The **Sender** property returns the address of the user who originated the message. Setting this property updates the X-Sender header with the specified value.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.StoreCount Property

Gets the number of messages in the current storage file, not including deleted messages.

[Visual Basic]
```
Public ReadOnly Property StoreCount As Integer
```

[C#]
```
public int StoreCount {get;}
```

Property Value

An integer value which specifies the number of messages in the message store, not including those messages that have been marked for deletion.

Remarks

The **StoreCount** property returns the number of messages in the message store. It is important to note that does not count those messages which have been marked for deletion. This means that the value returned by this method will decrease as messages are deleted. To determine the total number of messages, including deleted messages, use the **StoreSize** property.

See Also

MailMessage Class | SocketTools Namespace | StoreIndex Property | StoreSize Property

MailMessage.StoreFile Property

Gets and sets the name of the file used to store messages.

[Visual Basic]

```
Public Property StoreFile As String
```

[C#]

```
public string StoreFile {get; set;}
```

Property Value

A string value which specifies the name of the storage file.

Remarks

The **StoreFile** property returns the name of the current storage file. Setting this property changes the default filename that is used when opening a new storage file.

See Also

MailMessage Class | SocketTools Namespace | StoreCount Property | StoreIndex Property | StoreSize Property | OpenStore Method

MailMessage.StoreIndex Property

Gets and sets the current message index for the current storage file.

[Visual Basic]

```
Public Property StoreIndex As Integer
```

[C#]

```
public int StoreIndex {get; set;}
```

Property Value

An integer value which specifies the current message index into the storage file.

Remarks

The **StoreIndex** property returns the current message index for the message store. Setting this property changes the current message index. When no storage file has been opened, this property will return a value of zero. After a storage file has been opened, it is changed to a value of one, the first message in the message store. The maximum value for this property is the number of messages in the store, as returned by the **StoreSize** property. Attempting to set this property to a value less than one or greater than the number of messages in the store will result in an exception being thrown.

This property value is updated whenever the **ReadStore** or **ReplaceStore** methods are used. When the **WriteStore** method is used to store the current message in the message store, this property will be updated to reflect the message index of the newly added message.

See Also

MailMessage Class | SocketTools Namespace | StoreCount Property | StoreSize Property

MailMessage.StoreSize Property

Gets the total number of messages in the current storage file, including deleted messages.

[Visual Basic]

```
Public ReadOnly Property StoreSize As Integer
```

[C#]

```
public int StoreSize {get;}
```

Property Value

An integer value which specifies the total number of messages in the storage file.

Remarks

The **StoreSize** property returns the total number of messages in the message store, including those messages that have been deleted. Because the **StoreCount** property value will decrease as messages are deleted, it is recommended that you use this property value when iterating through all of the messages in the message store.

See Also

MailMessage Class | SocketTools Namespace | StoreCount Property | StoreIndex Property

MailMessage.Subject Property

Gets and sets the subject of the current message.

[Visual Basic]

```
Public Property Subject As String
```

[C#]

```
public string Subject {get; set;}
```

Property Value

A string which specifies the subject of the message.

Remarks

The **Subject** property returns the subject of the current message. Setting this property updates the Subject header with the specified value. Note that not all messages have subjects, in which case this property will be set to an empty string.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Text Property

Gets and sets the text body of the current message part.

[Visual Basic]
```
Public Property Text As String
```

[C#]
```
public string Text {get; set;}
```

Property Value

A string which contains the body of the current message part.

Remarks

The **Text** property returns the body of the current message part. Setting this property replaces the body of the current message part with the new text.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]

```
Public Property ThrowError As Boolean
```

[C#]

```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.TimeZone Property

Gets and sets the current timezone offset in seconds.

[Visual Basic]

```
Public Property TimeZone As Integer
```

[C#]

```
public int TimeZone {get; set;}
```

Property Value

An integer value which specifies the current timezone offset.

Remarks

The **TimeZone** property returns the current offset from UTC in seconds. Setting the property changes the current timezone offset to the specified value. The value of this property is initially determined by the date and time settings on the local system.

The **TimeZone** property value is used in conjunction with the **Localize** property to control how message date and time localization is handled.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.To Property

Gets and sets the address of the message recipient.

[Visual Basic]

```
Public Property To As String
```

[C#]

```
public string To {get; set;}
```

Property Value

A string which specifies the recipient of the message.

Remarks

The **To** property returns the address of the message recipient. Setting this property causes the To header to be updated with the specified value. Multiple addresses can be specified by separating them with commas.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Version Property

Gets a value which returns the current version of the MailMessage class library.

[Visual Basic]

```
Public ReadOnly Property Version As String
```

[C#]

```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the MailMessage class library. This value can be used by an application for validation and debugging purposes.

See Also

MailMessage Class | SocketTools Namespace

MailMessage Methods

The methods of the **MailMessage** class are listed below. For a complete list of **MailMessage** class members, see the MailMessage Members topic.

Public Instance Methods

AppendMessage	Append text to the body of the current message part.
AttachData	Overloaded. Attach the contents of a byte array to the current message.
AttachFile	Overloaded. Attach the specified file to the current message.
AttachImage	Overloaded. Attach an inline image to the current message.
AttachThread	Obsolete. Attach an instance of the class to the current thread.
ClearMessage	Clear the header and body of the current message.
CloseStore	Close the current message storage file.
ComposeMessage	Overloaded. Compose a new mail message.
CreatePart	Overloaded. Create a new message part in a multipart message.
DecodeText	Overloaded. Decode a string which was previously encoded using base64 or quoted-printable encoding.
DeleteHeader	Overloaded. Delete a header field from the specified message part.
DeleteMessage	Overloaded. Remove the specified message from the current message store.
DeletePart	Delete the specified message part from the current message.
Dispose	Overloaded. Releases all resources used by MailMessage.
EncodeText	Overloaded. Encodes a string using base64 or quoted-printable encoding.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
ExportMessage	Overloaded. Export the current message to a file on the local system.
ExtractAllFiles	Overloaded. Extract all file attachments from the current message.
ExtractFile	Overloaded. Extract the contents of a file

	attachment and store it on the local system.
FindAttachment	Search for a specific file attachment in the current message.
FindMessage	Overloaded. Search for a message in the current message store.
GetFirstHeader	Return the first header in the current message part.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetHeader	Overloaded. Return the value of a header field in the specified message part.
GetNextHeader	Return the next header in the current message part.
GetType (inherited from Object)	Gets the Type of the current instance.
ImportMessage	Replace the current message with the contents of a file.
Initialize	Overloaded. Initialize an instance of the MailMessage class.
OpenStore	Overloaded. Open the specified message storage file.
ParseAddress	Overloaded. Parse an Internet email address.
ParseMessage	Parse the specified string, adding the contents to the current message.
PurgeStore	Purge all deleted messages from the current message store.
ReadStore	Overloaded. Retrieve a message from the message store, replacing the current message.
ReplaceMessage	Overloaded. Replace the specified message in the current message store.
Reset	Reset the internal state of the object, resetting all properties to their default values.
SetHeader	Overloaded. Set the value for a header in the specified message part.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
WriteStore	Store the current message in the message store.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the MailMessage class and optionally

	releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the current message.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.AppendMessage Method

Append text to the body of the current message part.

[Visual Basic]

```
Public Function AppendMessage( _
   ByVal messageText As String _
) As Boolean
```

[C#]

```
public bool AppendMessage(
   string messageText
);
```

Parameters

messageText
A string which specifies the message text to be appended to the current message part.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.AttachData Method

Attach the contents of a byte array to the current message.

Overload List

Attach the contents of a byte array to the current message.

public bool AttachData(byte[],int);

Attach the contents of a byte array to the current message.

public bool AttachData(byte[],int,string);

Attach the contents of a byte array to the current message.

public bool AttachData(byte[],int,string,string);

Attach the contents of a byte array to the current message.

public bool AttachData(byte[],int,string,string,MimeAttachment);

Attach the contents of a string to the current message.

public bool AttachData(string);

Attach the contents of a string to the current message.

public bool AttachData(string,string);

Attach the contents of a string to the current message.

public bool AttachData(string,string,string);

Attach the contents of a string to the current message.

public bool AttachData(string,string,string,MimeAttachment);

See Also

MailMessage Class | SocketTools Namespace

MailMessage.AttachData Method (Byte[], Int32)

Attach the contents of a byte array to the current message.

[Visual Basic]

```
Overloads Public Function AttachData( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool AttachData(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array that contains the data to be attached to the message.

length

An integer value which specifies the maximum number of bytes top copy from the buffer. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The contents of the buffer will be attached to the message as inline content.

The buffer will be examined to determine what kind of data it contains. If there is only text characters, then the content type will be specified as text/plain. If the buffer contains binary data, then the content type will be specified as application/octet-stream, which is appropriate for any type of data.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.AttachData Overload List

MailMessage.AttachData Method (Byte[], Int32, String)

Attach the contents of a byte array to the current message.

[Visual Basic]

```
Overloads Public Function AttachData( _
   ByVal buffer As Byte(), _
   ByVal length As Integer, _
   ByVal contentName As String _
) As Boolean
```

[C#]

```
public bool AttachData(
   byte[] buffer,
   int length,
   string contentName
);
```

Parameters

buffer

A byte array that contains the data to be attached to the message.

length

An integer value which specifies the maximum number of bytes top copy from the buffer. This value cannot be larger than the size of the buffer specified by the caller.

contentName

An string argument which specifies a name for the data being attached to the message. This typically is used as a file name by the mail client to store the data in. If this parameter is omitted or passed as an empty string then no name is defined and the data is attached as inline content. Note that if a file name is specified with a path, only the base file name will be used.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The buffer will be examined to determine what kind of data it contains. If there is only text characters, then the content type will be specified as text/plain. If the buffer contains binary data, then the content type will be specified as application/octet-stream, which is appropriate for any type of data.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.AttachData Overload List

MailMessage.AttachData Method (Byte[], Int32, String, String)

Attach the contents of a byte array to the current message.

[Visual Basic]

```
Overloads Public Function AttachData( _
   ByVal buffer As Byte(), _
   ByVal length As Integer, _
   ByVal contentName As String, _
   ByVal contentType As String _
) As Boolean
```

[C#]

```
public bool AttachData(
   byte[] buffer,
   int length,
   string contentName,
   string contentType
);
```

Parameters

buffer

A byte array that contains the data to be attached to the message.

length

An integer value which specifies the maximum number of bytes top copy from the buffer. This value cannot be larger than the size of the buffer specified by the caller.

contentName

An string argument which specifies a name for the data being attached to the message. This typically is used as a file name by the mail client to store the data in. Note that if a file name is specified with a path, only the base file name will be used.

contentType

An string argument which specifies the type of data being attached. The value must be a valid MIME content type. If the buffer contains only text characters, then the content type will be specified as text/plain. If the buffer contains binary data, then the content type will be specified as application/octet-stream, which is appropriate for any type of data.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.AttachData Overload List

MailMessage.AttachData Method (Byte[], Int32, String, String, MimeAttachment)

Attach the contents of a byte array to the current message.

[Visual Basic]

```
Overloads Public Function AttachData( _
   ByVal buffer As Byte(), _
   ByVal length As Integer, _
   ByVal contentName As String, _
   ByVal contentType As String, _
   ByVal options As MimeAttachment _
) As Boolean
```

[C#]

```
public bool AttachData(
   byte[] buffer,
   int length,
   string contentName,
   string contentType,
   MimeAttachment options
);
```

Parameters

buffer

A byte array that contains the data to be attached to the message.

length

An integer value which specifies the maximum number of bytes top copy from the buffer. This value cannot be larger than the size of the buffer specified by the caller.

contentName

An string argument which specifies a name for the data being attached to the message. This typically is used as a file name by the mail client to store the data in. Note that if a file name is specified with a path, only the base file name will be used.

contentType

An string argument which specifies the type of data being attached. The value must be a valid MIME content type. If the buffer contains only text characters, then the content type will be specified as text/plain. If the buffer contains binary data, then the content type will be specified as application/octet-stream, which is appropriate for any type of data.

options

A MimeAttachment enumeration which specifies how the data should be attached to the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.AttachData Overload List

MailMessage.AttachData Method (String)

Attach the contents of a string to the current message.

[Visual Basic]

```
Overloads Public Function AttachData( _
   ByVal buffer As String _
) As Boolean
```

[C#]

```
public bool AttachData(
   string buffer
);
```

Parameters

buffer

A string that contains the data to be attached to the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The contents of the buffer will be attached to the message as inline content with a content type of text/plain.

This implementation of the method should never be used to attach binary data to a message. If you need to attach binary data, use the implementation of that accepts a byte array as the ***buffer*** parameter.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.AttachData Overload List

MailMessage.AttachData Method (String, String)

Attach the contents of a string to the current message.

[Visual Basic]

```
Overloads Public Function AttachData( _
   ByVal buffer As String, _
   ByVal contentName As String _
) As Boolean
```

[C#]

```
public bool AttachData(
   string buffer,
   string contentName
);
```

Parameters

buffer
A string that contains the data to be attached to the message.

contentName
An string argument which specifies a name for the data being attached to the message. This typically is used as a file name by the mail client to store the data in. Note that if a file name is specified with a path, only the base file name will be used.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The content will be attached using the content type of text/plain.

This implementation of the method should never be used to attach binary data to a message. If you need to attach binary data, use the implementation of that accepts a byte array as the ***buffer*** parameter.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.AttachData Overload List

MailMessage.AttachData Method (String, String, String)

Attach the contents of a string to the current message.

[Visual Basic]

```
Overloads Public Function AttachData( _
   ByVal buffer As String, _
   ByVal contentName As String, _
   ByVal contentType As String _
) As Boolean
```

[C#]

```
public bool AttachData(
   string buffer,
   string contentName,
   string contentType
);
```

Parameters

buffer
A string that contains the data to be attached to the message.

contentName
An string argument which specifies a name for the data being attached to the message. This typically is used as a file name by the mail client to store the data in.Note that if a file name is specified with a path, only the base file name will be used.

contentType
An string argument which specifies the type of data being attached. The value must be a valid MIME content type. If the buffer contains only text characters, then the content type will be specified as "text/plain". If the buffer contains binary data, then the content type will be specified as "application/octet-stream", which is appropriate for any type of data.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This implementation of the method should never be used to attach binary data to a message. If you need to attach binary data, use the implementation of that accepts a byte array as the ***buffer*** parameter.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.AttachData Overload List

MailMessage.AttachData Method (String, String, String, MimeAttachment)

Attach the contents of a string to the current message.

[Visual Basic]

```
Overloads Public Function AttachData( _
   ByVal buffer As String, _
   ByVal contentName As String, _
   ByVal contentType As String, _
   ByVal options As MimeAttachment _
) As Boolean
```

[C#]

```
public bool AttachData(
   string buffer,
   string contentName,
   string contentType,
   MimeAttachment options
);
```

Parameters

buffer
A string that contains the data to be attached to the message.

contentName
An string argument which specifies a name for the data being attached to the message. This typically is used as a file name by the mail client to store the data in.Note that if a file name is specified with a path, only the base file name will be used.

contentType
An string argument which specifies the type of data being attached. The value must be a valid MIME content type. If the buffer contains only text characters, then the content type will be specified as text/plain. If the buffer contains binary data, then the content type will be specified as application/octet-stream, which is appropriate for any type of data.

options
A MimeAttachment enumeration which specifies how the data should be attached to the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This implementation of the method should never be used to attach binary data to a message. If you need to attach binary data, use the implementation of that accepts a byte array as the ***buffer*** parameter.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.AttachData Overload List

MailMessage.AttachFile Method

Attach the specified file to the current message.

Overload List

Attach the specified file to the current message.

public bool AttachFile(string);

Attach the specified file to the current message.

public bool AttachFile(string,MimeAttachment);

See Also

MailMessage Class | SocketTools Namespace

MailMessage.AttachFile Method (String)

Attach the specified file to the current message.

```
[Visual Basic]
Overloads Public Function AttachFile( _
   ByVal fileName As String _
) As Boolean
```

```
[C#]
public bool AttachFile(
   string fileName
);
```

Parameters

fileName

A string which specifies the name of the file to be attached to the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AttachFile** method attaches the specified file to the current message. If the message already contains one or more file attachments, then it is added to the end of the message. If the message does not contain any attached files, then it is converted to a multipart message and the file is appended to the message.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.AttachFile Overload List

MailMessage.AttachFile Method (String, MimeAttachment)

Attach the specified file to the current message.

[Visual Basic]

```
Overloads Public Function AttachFile( _
   ByVal fileName As String, _
   ByVal options As MimeAttachment _
) As Boolean
```

[C#]

```
public bool AttachFile(
   string fileName,
   MimeAttachment options
);
```

Parameters

fileName

A string which specifies the name of the file to be attached to the message.

options

A MimeAttachment enumeration which specifies how the data should be attached to the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AttachFile** method attaches the specified file to the current message. If the message already contains one or more file attachments, then it is added to the end of the message. If the message does not contain any attached files, then it is converted to a multipart message and the file is appended to the message.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.AttachFile Overload List

MailMessage.AttachThread Method

NOTE: This method is now obsolete.

The AttachThread method has been deprecated

Attach an instance of the class to the current thread.

[Visual Basic]

```
<Obsolete(Message:="The AttachThread method has been deprecated", IsError:=False)>
Public Function AttachThread() As Boolean
```

[C#]

```
[Obsolete(Message="The AttachThread method has been deprecated", IsError=False)]
public bool AttachThread();
```

Return Value

A boolean value which specifies if the message could be attached to the current thread. If this method returns **false**, the message could not be attached to the thread and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method has been deprecated and should no longer be used. The current version of the MailMessage class uses a free threading model which permits any thread to access methods and properties. However, applications must take care to synchronize access to the class instance across multiple threads if they are modifying the message contents.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.ClearMessage Method

Clear the header and body of the current message.

[Visual Basic]

```
Public Function ClearMessage() As Boolean
```

[C#]

```
public bool ClearMessage();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ClearMessage** method clears the current message, releasing the memory allocated for the message and any attachments.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.CloseStore Method

Close the current message storage file.

[Visual Basic]

```
Public Function CloseStore() As Boolean
```

[C#]

```
public bool CloseStore();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CloseStore** method closes the storage file that was previously opened, releasing all of the memory allocated for the message store and purging all deleted messages. This method must be called when the application has finished accessing the messages in the message store.

When the control instance is released by its container, the storage file will automatically be closed. To prevent deleted messages from being removed from the message store, use the **Reset** method.

See Also

MailMessage Class | SocketTools Namespace | OpenStore Method

MailMessage.ComposeMessage Method

Compose a new mail message.

Overload List

Compose a new mail message.

public bool ComposeMessage(string,string,string,string);

Compose a new mail message.

public bool ComposeMessage(string,string,string,string,string);

Compose a new mail message.

public bool ComposeMessage(string,string,string,string,string,string);

Compose a new mail message.

public bool ComposeMessage(string,string,string,string,string,string,string);

Compose a new mail message.

public bool ComposeMessage(string,string,string,string,string,string,string,MimeCharacterSet,MimeEncoding);

See Also

MailMessage Class | SocketTools Namespace

MailMessage.ComposeMessage Method (String, String, String, String)

Compose a new mail message.

[Visual Basic]

```
Overloads Public Function ComposeMessage( _
   ByVal messageFrom As String, _
   ByVal messageTo As String, _
   ByVal messageSubject As String, _
   ByVal messageText As String _
) As Boolean
```

[C#]

```
public bool ComposeMessage(
   string messageFrom,
   string messageTo,
   string messageSubject,
   string messageText
);
```

Parameters

messageFrom
A string argument which specifies the sender's email address. Only a single address should be specified. After the message has been composed, the **From** property will be updated with this value.

messageTo
A string argument which specifies one or more recipient email addresses. Multiple email addresses may be specified by separating them with commas. After the message has been composed, the **To** property will be updated with this value.

messageSubject
A string argument which specifies the subject for the message. If the argument is not specified, then no Subject header field will be created for this message. After the message has been composed, the **Subject** property will be updated with this value

messageText
An string argument which specifies the body of the message. Each line of text contained in the string should be terminated with a carriage-return/linefeed (CRLF) pair, which is recognized as the end-of-line.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ComposeMessage** method creates a new mail message, or replaces the current message if one already exists.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.ComposeMessage Overload List

MailMessage.ComposeMessage Method (String, String, String, String, String)

Compose a new mail message.

[Visual Basic]

```
Overloads Public Function ComposeMessage( _
   ByVal messageFrom As String, _
   ByVal messageTo As String, _
   ByVal messageCc As String, _
   ByVal messageSubject As String, _
   ByVal messageText As String _
) As Boolean
```

[C#]

```
public bool ComposeMessage(
   string messageFrom,
   string messageTo,
   string messageCc,
   string messageSubject,
   string messageText
);
```

Parameters

messageFrom
A string argument which specifies the sender's email address. Only a single address should be specified. After the message has been composed, the **From** property will be updated with this value.

messageTo
A string argument which specifies one or more recipient email addresses. Multiple email addresses may be specified by separating them with commas. After the message has been composed, the **To** property will be updated with this value.

messageCc
A string argument which specifies one or more additional recipient addresses that will receive a copy of the message. If this argument is not specified, then no Cc header field will be created for this message. After the message has been composed, the **Cc** property will be updated with this value.

messageSubject
A string argument which specifies the subject for the message. If the argument is not specified, then no Subject header field will be created for this message. After the message has been composed, the **Subject** property will be updated with this value

messageText
An string argument which specifies the body of the message. Each line of text contained in the string should be terminated with a carriage-return/linefeed (CRLF) pair, which is recognized as the end-of-line.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ComposeMessage** method creates a new mail message, or replaces the current message if one already exists.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.ComposeMessage Overload List

MailMessage.ComposeMessage Method (String, String, String, String, String, String)

Compose a new mail message.

[Visual Basic]

```
Overloads Public Function ComposeMessage( _
   ByVal messageFrom As String, _
   ByVal messageTo As String, _
   ByVal messageCc As String, _
   ByVal messageBcc As String, _
   ByVal messageSubject As String, _
   ByVal messageText As String _
) As Boolean
```

[C#]

```
public bool ComposeMessage(
   string messageFrom,
   string messageTo,
   string messageCc,
   string messageBcc,
   string messageSubject,
   string messageText
);
```

Parameters

messageFrom

A string argument which specifies the sender's email address. Only a single address should be specified. After the message has been composed, the **From** property will be updated with this value.

messageTo

A string argument which specifies one or more recipient email addresses. Multiple email addresses may be specified by separating them with commas. After the message has been composed, the **To** property will be updated with this value.

messageCc

A string argument which specifies one or more additional recipient addresses that will receive a copy of the message. If this argument is not specified, then no Cc header field will be created for this message. After the message has been composed, the **Cc** property will be updated with this value.

messageBcc

A string argument which specifies one or more additional recipient addresses that will receive a "blind" copy of the message. If this argument is not specified, then no Bcc header field will be created for this message. After the message has been composed, the **Bcc** property will be updated with this value. Note that the Bcc header field is not normally included in the header when the message is exported.

messageSubject

A string argument which specifies the subject for the message. If the argument is not specified, then no Subject header field will be created for this message. After the message has been composed, the **Subject** property will be updated with this value

messageText

An string argument which specifies the body of the message. Each line of text contained in the string should be terminated with a carriage-return/linefeed (CRLF) pair, which is recognized as the end-of-line.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ComposeMessage** method creates a new mail message, or replaces the current message if one already exists.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.ComposeMessage Overload List

MailMessage.ComposeMessage Method (String, String, String, String, String, String, String)

Compose a new mail message.

[Visual Basic]

```
Overloads Public Function ComposeMessage( _
   ByVal messageFrom As String, _
   ByVal messageTo As String, _
   ByVal messageCc As String, _
   ByVal messageBcc As String, _
   ByVal messageSubject As String, _
   ByVal messageText As String, _
   ByVal messageHTML As String _
) As Boolean
```

[C#]

```
public bool ComposeMessage(
   string messageFrom,
   string messageTo,
   string messageCc,
   string messageBcc,
   string messageSubject,
   string messageText,
   string messageHTML
);
```

Parameters

messageFrom

A string argument which specifies the sender's email address. Only a single address should be specified. After the message has been composed, the **From** property will be updated with this value.

messageTo

A string argument which specifies one or more recipient email addresses. Multiple email addresses may be specified by separating them with commas. After the message has been composed, the **To** property will be updated with this value.

messageCc

A string argument which specifies one or more additional recipient addresses that will receive a copy of the message. If this argument is not specified, then no Cc header field will be created for this message. After the message has been composed, the **Cc** property will be updated with this value.

messageBcc

A string argument which specifies one or more additional recipient addresses that will receive a "blind" copy of the message. If this argument is not specified, then no Bcc header field will be created for this message. After the message has been composed, the **Bcc** property will be updated with this value. Note that the Bcc header field is not normally included in the header when the message is exported.

messageSubject

A string argument which specifies the subject for the message. If the argument is not specified, then no Subject header field will be created for this message. After the message has been composed, the **Subject** property will be updated with this value

messageText

An string argument which specifies the body of the message. Each line of text contained in the string should be terminated with a carriage-return/linefeed (CRLF) pair, which is recognized as the end-of-line.

messageHTML

A string argument which specifies an alternate HTML formatted message. If the ***messageText*** argument has been specified, then a multipart message will be created with both plain text and HTML text as the alternative. This allows mail clients to select which message body they wish to display. If the ***messageText*** argument is an empty string, then the message will only contain HTML. Although this is supported, it is not recommended because older mail clients may be unable to display the message correctly.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ComposeMessage** method creates a new mail message, or replaces the current message if one already exists.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.ComposeMessage Overload List

MailMessage.ComposeMessage Method (String, String, String, String, String, String, String, MimeCharacterSet, MimeEncoding)

Compose a new mail message.

[Visual Basic]

```
Overloads Public Function ComposeMessage( _
   ByVal messageFrom As String, _
   ByVal messageTo As String, _
   ByVal messageCc As String, _
   ByVal messageBcc As String, _
   ByVal messageSubject As String, _
   ByVal messageText As String, _
   ByVal messageHTML As String, _
   ByVal characterSet As MimeCharacterSet, _
   ByVal encodingType As MimeEncoding _
) As Boolean
```

[C#]

```
public bool ComposeMessage(
   string messageFrom,
   string messageTo,
   string messageCc,
   string messageBcc,
   string messageSubject,
   string messageText,
   string messageHTML,
   MimeCharacterSet characterSet,
   MimeEncoding encodingType
);
```

Parameters

messageFrom

A string argument which specifies the sender's email address. Only a single address should be specified. After the message has been composed, the **From** property will be updated with this value.

messageTo

A string argument which specifies one or more recipient email addresses. Multiple email addresses may be specified by separating them with commas. After the message has been composed, the **To** property will be updated with this value.

messageCc

A string argument which specifies one or more additional recipient addresses that will receive a copy of the message. If this argument is not specified, then no Cc header field will be created for this message. After the message has been composed, the **Cc** property will be updated with this value.

messageBcc

A string argument which specifies one or more additional recipient addresses that will receive a "blind" copy of the message. If this argument is not specified, then no Bcc header field will be created for this message. After the message has been composed, the **Bcc** property will be updated with this value. Note that the Bcc header field is not normally included in the header when the message is exported.

messageSubject

A string argument which specifies the subject for the message. If the argument is not specified, then no Subject header field will be created for this message. After the message has been composed, the **Subject** property will be updated with this value *messageText*

An string argument which specifies the body of the message. Each line of text contained in the string should be terminated with a carriage-return/linefeed (CRLF) pair, which is recognized as the end-of-line.

messageHTML

A string argument which specifies an alternate HTML formatted message. If the ***messageText*** argument has been specified, then a multipart message will be created with both plain text and HTML text as the alternative. This allows mail clients to select which message body they wish to display. If the ***messageText*** argument is an empty string, then the message will only contain HTML. Although this is supported, it is not recommended because older mail clients may be unable to display the message correctly.

characterSet

A MimeCharacterSet enumeration value which specifies the character set to use when composing the message.

encodingType

A MimeEncoding enumeration value which specifies the encoding type to use when composing the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ComposeMessage** method creates a new mail message, or replaces the current message if one already exists.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.ComposeMessage Overload List

MailMessage.CreatePart Method

Create an empty message part in a multipart message.

Overload List

Create an empty message part in a multipart message.

public bool CreatePart();

Create a new message part in a multipart message.

public bool CreatePart(string);

Create a new message part in a multipart message.

public bool CreatePart(string,MimeCharacterSet,MimeEncoding);

See Also

MailMessage Class | SocketTools Namespace

MailMessage.CreatePart Method ()

Create an empty message part in a multipart message.

[Visual Basic]

```
Overloads Public Function CreatePart() As Boolean
```

[C#]

```
public bool CreatePart();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreatePart** method creates a new empty message part. If the current message is a simple RFC822 formatted message, then this method converts it to a MIME multipart message. The current message part will be set to the new part that was just created.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.CreatePart Overload List

MailMessage.CreatePart Method (String)

Create a new message part in a multipart message.

[Visual Basic]

```
Overloads Public Function CreatePart( _
   ByVal messageText As String _
) As Boolean
```

[C#]

```
public bool CreatePart(
   string messageText
);
```

Parameters

messageText

A string argument which specifies the body of the new message part. Each line of text contained in the string should be terminated with a carriage-return/linefeed (CRLF) pair, which is recognized as the end-of-line.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreatePart** method creates a new message part. If the current message is a simple RFC822 formatted message, then this method converts it to a MIME multipart message. The current message part will be set to the new part that was just created.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.CreatePart Overload List

MailMessage.CreatePart Method (String, MimeCharacterSet, MimeEncoding)

Create a new message part in a multipart message.

[Visual Basic]

```
Overloads Public Function CreatePart( _
   ByVal messageText As String, _
   ByVal characterSet As MimeCharacterSet, _
   ByVal encodingType As MimeEncoding _
) As Boolean
```

[C#]

```
public bool CreatePart(
   string messageText,
   MimeCharacterSet characterSet,
   MimeEncoding encodingType
);
```

Parameters

messageText
A string argument which specifies the body of the new message part. Each line of text contained in the string should be terminated with a carriage-return/linefeed (CRLF) pair, which is recognized as the end-of-line.

characterSet
A MimeCharacterSet enumeration value which specifies the character set to use when composing the message.

encodingType
A MimeEncoding enumeration value which specifies the encoding type to use when composing the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreatePart** method creates a new message part. If the current message is a simple RFC822 formatted message, then this method converts it to a MIME multipart message. The current message part will be set to the new part that was just created.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.CreatePart Overload List

MailMessage.DecodeText Method

Decode a string which was previously encoded using base64 or quoted-printable encoding.

Overload List

Decode a string which was previously encoded using base64 or quoted-printable encoding.

public int DecodeText(string,ref string);

Decode a string which was previously encoded using base64 or quoted-printable encoding.

public int DecodeText(string,ref string,MimeCharacterSet,MimeEncoding);

Decode a string which was previously encoded using base64 or quoted-printable encoding.

public int DecodeText(string,ref string,MimeEncoding);

See Also

MailMessage Class | SocketTools Namespace

MailMessage.DecodeText Method (String, String, MimeCharacterSet, MimeEncoding)

Decode a string which was previously encoded using base64 or quoted-printable encoding.

[Visual Basic]

```
Overloads Public Function DecodeText( _
   ByVal encodedText As String, _
   ByRef messageText As String, _
   ByVal charSet As MimeCharacterSet, _
   ByVal encoding As MimeEncoding _
) As Integer
```

[C#]

```
public int DecodeText(
   string encodedText,
   ref string messageText,
   MimeCharacterSet charSet,
   MimeEncoding encoding
);
```

Parameters

encodedText
A string which contains the encoded text which should be decoded.

messageText
A string variable passed by reference which will contain the decoded text when the method returns.

charSet
A MimeCharacterSet enumeration which specifies the character set to use when decoding the encoded text. If this value does not match the character set used when the text was originally encoded, the resulting output text may be invalid.

encoding
A MimeEncoding enumeration which specifies the content encoding type to use when decoding the text.

Return Value

The number of characters of decoded text. A return value of zero indicates no text has been decoded. If the method fails, it will return -1 and the LastError property can be used to determine the cause of the failure. In most cases where the method fails, it is because an invalid character set or encoding type has been specified.

Remarks

This method provides a means to decode text that was previously encoded using either base64 or quoted-printable encoding. In most cases, it is not necessary to use this method because the message parser will detect which character set and encoding was used, then automatically decode the message text if necessary.

The value of the ***charSet*** parameter does not affect the resulting output text, it is only used when decoding the input text. The previous contents of the ***messageText*** string will be replaced by the decoded text, and the output string will always be Unicode.

If the ***charSet*** parameter is specified as **charsetUTF16**, the encoding type must be **encodingBase64**. Other encoding methods are not supported for Unicode strings and will cause the method to fail. In most cases, it is preferable to always use **encodingBase64** as the encoding method, with quoted-printable encoding only used for legacy support. If an unsupported encoding type is specified, this method will return -1 and the output text string will be empty. This method cannot be used to decode uuencoded text.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.DecodeText Overload List

MailMessage.DecodeText Method (String, String, MimeEncoding)

Decode a string which was previously encoded using base64 or quoted-printable encoding.

[Visual Basic]

```
Overloads Public Function DecodeText( _
   ByVal encodedText As String, _
   ByRef messageText As String, _
   ByVal encoding As MimeEncoding _
) As Integer
```

[C#]

```
public int DecodeText(
   string encodedText,
   ref string messageText,
   MimeEncoding encoding
);
```

Parameters

encodedText
A string which contains the encoded text which should be decoded.

messageText
A string variable passed by reference which will contain the decoded text when the method returns.

encoding
A MimeEncoding enumeration which specifies the content encoding type to use when decoding the text.

Return Value

The number of characters of decoded text. A return value of zero indicates no text has been decoded. If the method fails, it will return -1 and the LastError property can be used to determine the cause of the failure. In most cases where the method fails, it is because an invalid character set or encoding type has been specified.

Remarks

This method provides a means to decode text that was previously encoded using either base64 or quoted-printable encoding. In most cases, it is not necessary to use this method because the message parser will detect which character set and encoding was used, then automatically decode the message text if necessary.

The value of the ***charSet*** parameter does not affect the resulting output text, it is only used when decoding the input text. The previous contents of the ***messageText*** string will be replaced by the decoded text, and the output string will always be Unicode.

In most cases, it is preferable to always use **encodingBase64** as the encoding method, with quoted-printable encoding only used for legacy support. If an unsupported encoding type is specified, this method will return -1 and the output text string will be empty.

This version of the method will default to decoding the text as UTF-8. If the original text used something other than UTF-8, call the overloaded version of this method to specify the character set.

This method cannot be used to deocde uuencoded text.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.DecodeText Overload List

MailMessage.DecodeText Method (String, String)

Decode a string which was previously encoded using base64 or quoted-printable encoding.

[Visual Basic]

```
Overloads Public Function DecodeText( _
   ByVal encodedText As String, _
   ByRef messageText As String _
) As Integer
```

[C#]

```
public int DecodeText(
   string encodedText,
   ref string messageText
);
```

Parameters

encodedText
A string which contains the encoded text which should be decoded.

messageText
A string variable passed by reference which will contain the decoded text when the method returns.

Return Value

The number of characters of decoded text. A return value of zero indicates no text has been decoded. If the method fails, it will return -1 and the LastError property can be used to determine the cause of the failure. In most cases where the method fails, it is because an invalid character set or encoding type has been specified.

Remarks

This method provides a means to decode text that was previously encoded using base64 encoding. In most cases, it is not necessary to use this method because the message parser will detect which character set and encoding was used, then automatically decode the message text if necessary.

This version of the method will default to using base64 encoding and will decode the text as UTF-8. If the original text used something other than UTF-8, call the overloaded version of this method to specify the character set.

This method cannot be used to deocde uuencoded text.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.DecodeText Overload List

MailMessage.DeleteHeader Method

Delete a header field from the specified message part.

Overload List

Delete a header field from the specified message part.

public bool DeleteHeader(int,string);

Delete a header field from the current message part.

public bool DeleteHeader(string);

See Also

MailMessage Class | SocketTools Namespace

MailMessage.DeleteHeader Method (Int32, String)

Delete a header field from the specified message part.

[Visual Basic]

```
Overloads Public Function DeleteHeader( _
   ByVal messagePart As Integer, _
   ByVal headerName As String _
) As Boolean
```

[C#]

```
public bool DeleteHeader(
   int messagePart,
   string headerName
);
```

Parameters

messagePart
An integer which specifies the message part.

headerName
A string which specifies the header field to delete from the specified message part.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

A message part of zero specifies the main message part which contains the standard headers such as To, From and Subject. The number of message parts in the current message is returned by the **PartCount** property.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.DeleteHeader Overload List

MailMessage.DeleteHeader Method (String)

Delete a header field from the current message part.

[Visual Basic]

```
Overloads Public Function DeleteHeader( _
   ByVal headerName As String _
) As Boolean
```

[C#]

```
public bool DeleteHeader(
   string headerName
);
```

Parameters

headerName

A string which specifies the header field to delete from the current message part.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The current message part is returned by the **Part** property.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.DeleteHeader Overload List

MailMessage.DeleteMessage Method

Remove the current message from the current message store.

Overload List

Remove the current message from the current message store.

public bool DeleteMessage();

Remove the specified message from the current message store.

public bool DeleteMessage(int);

See Also

MailMessage Class | SocketTools Namespace

MailMessage.DeleteMessage Method (Int32)

Remove the specified message from the current message store.

[Visual Basic]

```
Overloads Public Function DeleteMessage( _
   ByVal messageIndex As Integer _
) As Boolean
```

[C#]

```
public bool DeleteMessage(
   int messageIndex
);
```

Parameters

messageIndex

An integer value which specifies the message that will be removed from the message store.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **DeleteMessage** method marks the specified message for deletion from the storage file. When the message store is closed or purged, the message is removed from the file.

Once a message has been marked for deletion, it may no longer be referenced by the application. For example, you cannot access the contents of a message that has been deleted.

The message store must be opened with write access. This method will fail if you attempt to delete a message from a storage file that has been opened for read-only access. If the application needs to delete messages in the message store, it is recommended that the file be opened for exclusive access using the **storeLock** option when calling the **OpenStore** method.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.DeleteMessage Overload List

MailMessage.DeleteMessage Method ()

Remove the current message from the current message store.

[Visual Basic]

```
Overloads Public Function DeleteMessage() As Boolean
```

[C#]

```
public bool DeleteMessage();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **DeleteMessage** method marks the current message for deletion from the storage file. When the message store is closed or purged, the message is removed from the file. The value of the **StoreIndex** property specifies the current message.

Once a message has been marked for deletion, it may no longer be referenced by the application. For example, you cannot access the contents of a message that has been deleted.

The message store must be opened with write access. This method will fail if you attempt to delete a message from a storage file that has been opened for read-only access. If the application needs to delete messages in the message store, it is recommended that the file be opened for exclusive access using the **storeLock** option when calling the **OpenStore** method.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.DeleteMessage Overload List

MailMessage.DeletePart Method

Delete the specified message part from the current message.

[Visual Basic]

```
Public Function DeletePart( _
   ByVal messagePart As Integer _
) As Boolean
```

[C#]

```
public bool DeletePart(
   int messagePart
);
```

Parameters

messagePart

An integer which specifies the message part.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method cannot be used to delete part zero, which is the main body of the message. Instead use the **ClearMessage** method to clear the contents of the entire message.

The number of message parts in the current message is returned by the **PartCount** property.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Dispose Method

Releases all resources used by MailMessage.

Overload List

Releases all resources used by MailMessage.

public void Dispose();

Releases the unmanaged resources allocated by the MailMessage class and optionally releases the managed resources.

protected virtual void Dispose(bool);

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Dispose Method ()

Releases all resources used by MailMessage.

[Visual Basic]
```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]
```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.Dispose Overload List

MailMessage.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the MailMessage class and optionally releases the managed resources.

[Visual Basic]

```
Overridable Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]

```
protected virtual void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **MailMessage** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.Dispose Overload List

MailMessage.EncodeText Method

Encodes a string using base64 or quoted-printable encoding.

Overload List

Encodes a string using base64 or quoted-printable encoding.

public int EncodeText(string,ref string);

Encodes a string using base64 or quoted-printable encoding.

public int EncodeText(string,ref string,MimeCharacterSet,MimeEncoding);

Encodes a string using base64 or quoted-printable encoding.

public int EncodeText(string,ref string,MimeEncoding);

See Also

MailMessage Class | SocketTools Namespace

MailMessage.EncodeText Method (String, String, MimeCharacterSet, MimeEncoding)

Encodes a string using base64 or quoted-printable encoding.

```
[Visual Basic]
Overloads Public Function EncodeText( _
   ByVal messageText As String, _
   ByRef encodedText As String, _
   ByVal charSet As MimeCharacterSet, _
   ByVal encoding As MimeEncoding _
) As Integer
```

```
[C#]
public int EncodeText(
   string messageText,
   ref string encodedText,
   MimeCharacterSet charSet,
   MimeEncoding encoding
);
```

Parameters

messageText
A string that contains the text which should be encoded.

encodedText
A string variable passed by reference which will contain the encoded text when the method returns.

charSet
A MimeCharacterSet enumeration which specifies the character set to use when encoding the message text. It is recommended that you use UTF-8 whenever possible.

encoding
A MimeEncoding enumeration which specifies the content encoding type to use when encoding the text.

Return Value

The number of characters of encoded text. A return value of zero indicates no text has been encoded. If the method fails, it will return -1 and the LastError property can be used to determine the cause of the failure. In most cases where the method fails, it is because an invalid character set or encoding type has been specified.

Remarks

This method provides a means to encode text using either base64 or quoted-printable encoding. It is not necessary to use this method to encode text when assigning a value to Text property. The class will automatically encode message text which contains non-ASCII characters using the character set specified when the message is created.

If the ***charSet*** parameter is specified, the method will convert the message text using the ANSI code page associated with the character set, and then the text will be encoded.

If the ***charsetUTF16*** character set is specified, you must also specify ***encodingBase64*** as the encoding method. Other encoding methods are not supported and this will cause the method to fail. It is not recommended you encode text as UTF-16 unless there is a specific requirement to use that character set.

It is recommended that you use the ***charsetUTF8*** character set whenever possible. It is capable of encoding all Unicode code points, and is a standard for virtually all modern Internet applications. In most cases, it is preferable to use ***encodingBase64*** as the encoding method, with quoted-printable encoding only used for legacy support.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.EncodeText Overload List

MailMessage.EncodeText Method (String, String, MimeEncoding)

Encodes a string using base64 or quoted-printable encoding.

[Visual Basic]

```
Overloads Public Function EncodeText( _
   ByVal messageText As String, _
   ByRef encodedText As String, _
   ByVal encoding As MimeEncoding _
) As Integer
```

[C#]

```
public int EncodeText(
   string messageText,
   ref string encodedText,
   MimeEncoding encoding
);
```

Parameters

messageText
A string that contains the text which will be encoded.

encodedText
A string variable passed by reference which will contain the encoded text when the method returns.

encoding
A MimeEncoding enumeration which specifies the content encoding type to use when decoding the text.

Return Value

The number of characters of encoded text. A return value of zero indicates no text has been encoded. If the method fails, it will return -1 and the LastError property can be used to determine the cause of the failure. In most cases where the method fails, it is because an invalid character set or encoding type has been specified.

Remarks

This method provides a means to encode text using either base64 or quoted-printable encoding. It is not necessary to use this method to encode text when assigning a value to the **Text** property. The class will automatically encode message text which contains non-ASCII characters using the character set specified when the message is created.

In most cases, it is preferable to use ***encodingBase64*** as the encoding method, with quoted-printable encoding only used for legacy support.

This version of the method will convert the message text to UTF-8 prior to encoding the text.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.EncodeText Overload List

MailMessage.EncodeText Method (String, String)

Encodes a string using base64 or quoted-printable encoding.

[Visual Basic]

```
Overloads Public Function EncodeText( _
   ByVal messageText As String, _
   ByRef encodedText As String _
) As Integer
```

[C#]

```
public int EncodeText(
   string messageText,
   ref string encodedText
);
```

Parameters

messageText
A string that contains the text which will be encoded.

encodedText
A string variable passed by reference which will contain the encoded text when the method returns.

Return Value

The number of characters of encoded text. A return value of zero indicates no text has been encoded. If the method fails, it will return -1 and the LastError property can be used to determine the cause of the failure. In most cases where the method fails, it is because an invalid character set or encoding type has been specified.

Remarks

This method provides a means to encode text using base64. It is not necessary to use this method to encode text when assigning a value to the **Text** property. The class will automatically encode message text which contains non-ASCII characters using the character set specified when the message is created.

This version of the method will convert the message text to UTF-8 and then encode it using the base64 algorithm.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.EncodeText Overload List

MailMessage.ExportMessage Method

Export the current message to a file on the local system.

Overload List

Export the current message to a file on the local system.

public bool ExportMessage(string);

Export the current message to a file on the local system.

public bool ExportMessage(string,MimeExportOptions);

See Also

MailMessage Class | SocketTools Namespace

MailMessage.ExportMessage Method (String)

Export the current message to a file on the local system.

[Visual Basic]

```
Overloads Public Function ExportMessage( _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool ExportMessage(
   string fileName
);
```

Parameters

fileName

A string which specifies the name of the file that will contain the message. If the file does not exist, it will be created. If it does exist, it will be overwritten with the contents of the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ExportMessage** method writes the current message to a file. If the file does not exist, it will be created. If it does exist, it will be overwritten with the contents of the message.

The value of the **Options** property determines the default export options, if any have been specified.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.ExportMessage Overload List

MailMessage.ExportMessage Method (String, MimeExportOptions)

Export the current message to a file on the local system.

[Visual Basic]

```
Overloads Public Function ExportMessage( _
   ByVal fileName As String, _
   ByVal options As MimeExportOptions _
) As Boolean
```

[C#]

```
public bool ExportMessage(
   string fileName,
   MimeExportOptions options
);
```

Parameters

fileName

A string which specifies the name of the file that will contain the message. If the file does not exist, it will be created. If it does exist, it will be overwritten with the contents of the message.

options

A MimeExportOptions enumeration value which specifies one or more export options.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ExportMessage** method writes the current message to a file. If the file does not exist, it will be created. If it does exist, it will be overwritten with the contents of the message.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.ExportMessage Overload List

MailMessage.ExtractAllFiles Method

Extract all file attachments from the current message.

Overload List

Extract all file attachments from the current message.

public int ExtractAllFiles();

Extract all file attachments from the current message.

public int ExtractAllFiles(string);

See Also

MailMessage Class | SocketTools Namespace

MailMessage.ExtractAllFiles Method ()

Extract all file attachments from the current message.

[Visual Basic]
```
Overloads Public Function ExtractAllFiles() As Integer
```

[C#]
```
public int ExtractAllFiles();
```

Return Value

This method returns an integer value. If the method succeeds, the return value is the number of attachments that were extracted from the message. A return value of zero indicates that the message did not contain any file attachments. If the method faile, the return value is -1. To get extended error information, check the value of the **LastError** property.

Remarks

This method will extract all of the files that are attached to the current message and store them in the current directory on the local system. If a file with the same name as the attachment already exists, it will be overwritten with the contents of the attachment. If the file attachment was encoded using base64 or uuencode, this method will automatically decode the contents of the attachment.

To store a file attachment on the local system using a name that is different than the file name of the attachment, use the **ExtractFile** method.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.ExtractAllFiles Overload List

MailMessage.ExtractAllFiles Method (String)

Extract all file attachments from the current message.

[Visual Basic]

```
Overloads Public Function ExtractAllFiles( _
   ByVal pathName As String _
) As Integer
```

[C#]

```
public int ExtractAllFiles(
   string pathName
);
```

Parameters

pathName

A string that specifies the name of the directory where the file attachments should be stored. If this parameter is omitted or points to an empty string, the attached files will be stored in the current working directory on the local system.

Return Value

This method returns an integer value. If the method succeeds, the return value is the number of attachments that were extracted from the message. A return value of zero indicates that the message did not contain any file attachments. If the method faile, the return value is -1. To get extended error information, check the value of the **LastError** property.

Remarks

This method will extract all of the files that are attached to the current message and store them in the specified directory. The directory must exist and the current user must have the appropriate permissions to create files there. If a file with the same name as the attachment already exists, it will be overwritten with the contents of the attachment. If the file attachment was encoded using base64 or uuencode, this method will automatically decode the contents of the attachment.

To store a file attachment on the local system using a name that is different than the file name of the attachment, use the **ExtractFile** method.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.ExtractAllFiles Overload List

MailMessage.ExtractFile Method

Extract the contents of a file attachment and store it on the local system.

Overload List

Extract the contents of a file attachment and store it on the local system.

public bool ExtractFile(int,string);

Extract the contents of a file attachment and store it on the local system.

public bool ExtractFile(string);

Extract the contents of a file attachment and store it on the local system.

public bool ExtractFile(string,string);

See Also

MailMessage Class | SocketTools Namespace

MailMessage.ExtractFile Method (Int32, String)

Extract the contents of a file attachment and store it on the local system.

[Visual Basic]

```
Overloads Public Function ExtractFile( _
   ByVal messagePart As Integer, _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool ExtractFile(
   int messagePart,
   string fileName
);
```

Parameters

messagePart

An integer which specifies the message part.

fileName

A string which specifies the name of the file that will contain the file attachment. If the file does not exist, it will be created. If it does exist, it will be overwritten with the contents of the attachment.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ExtractFile** method writes the contents of a message part, typically a file attachment, to a file on the local system. This method will automatically decode any binary file attachments.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.ExtractFile Overload List

MailMessage.ExtractFile Method (String, String)

Extract the contents of a file attachment and store it on the local system.

[Visual Basic]

```
Overloads Public Function ExtractFile( _
   ByVal attachName As String, _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool ExtractFile(
   string attachName,
   string fileName
);
```

Parameters

attachName

A string that specifies the name of the file attachment in the current message. This parameter should only specify a base file name; it should not include a file path and cannot be an empty string

fileName

A string which specifies the name of the file on the local system that will contain the file attachment. If the file does not exist, it will be created. If it does exist, it will be overwritten with the contents of the attachment.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This version of the **ExtractFile** method will search the current message for a file attachment that matches the name specified by the ***attachName*** parameter. If an attachment with that name is found, its contents will be stored in the local file specified by the ***fileName*** parameter. If the message does not contain an attachment that matches the name provided, this method will fail.

To search for a file attachment by name, use the **FindAttachment** method.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.ExtractFile Overload List

MailMessage.Finalize Method

Destroys an instance of the class, releasing the resources allocated for the current message.

[Visual Basic]

```
Overrides Protected Sub Finalize()
```

[C#]

```
protected override void Finalize();
```

See Also

MailMessage Class | SocketTools Namespace

MailMessage.FindAttachment Method

Search for a specific file attachment in the current message.

[Visual Basic]

```
Public Function FindAttachment( _
   ByVal fileName As String _
) As Integer
```

[C#]

```
public int FindAttachment(
   string fileName
);
```

Parameters

fileName

A string value that specifies the name of the file attachment to search for. This parameter should only specify a base file name; it should not include a file path and cannot be an empty string.

Return Value

An integer value which specifies the message part number that contains the file attachment with a matching name. If the message does not contain a file attachment with the specified name, this method will return -1.

Remarks

The **FindAttachment** method will search the current message for a attachment that matches the specified file name. The search is not case-sensitive, however it must match the attachment file name completely. This method will not match partial file names or names that include wildcard characters.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.FindMessage Method

Search for a message in the current message store.

Overload List

Search for a message in the current message store.

public int FindMessage(int,string,string);

Search for a message in the current message store.

public int FindMessage(int,string,string,MimeSearch);

Search for a message in the current message store.

public int FindMessage(string,string);

Search for a message in the current message store.

public int FindMessage(string,string,MimeSearch);

See Also

MailMessage Class | SocketTools Namespace

MailMessage.FindMessage Method (Int32, String, String, MimeSearch)

Search for a message in the current message store.

[Visual Basic]

```
Overloads Public Function FindMessage( _
   ByVal messageIndex As Integer, _
   ByVal headerField As String, _
   ByVal headerValue As String, _
   ByVal searchOptions As MimeSearch _
) As Integer
```

[C#]

```
public int FindMessage(
   int messageIndex,
   string headerField,
   string headerValue,
   MimeSearch searchOptions
);
```

Parameters

messageIndex
An integer value which specifies the message number that should be used when starting the search.

headerField
A string which specifies the name of the header field that should be searched. The header field name is not case sensitive.

headerValue
A string which specifies the header value that should be searched for. The search options can be used to specify if the search is case-sensitive, and whether the search should return partial matches to the string.

searchOptions
A MimeSearch enumeration which specifies the options to be used when searching for a message.

Return Value

An integer value which specifies the message index for the message that matched the search criteria. If no matching message was found, this method will return zero.

Remarks

The **FindMessage** method is used to search the message store for a message which matches a specific header field value. For example, it can be used to find every message which is addressed to a specific recipient or has a subject which matches a particular string value.

When searching for specific email addresses, it is recommended that you use the **searchPartialMatch** option because of the different ways that addresses can be annotated.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.FindMessage Overload List

MailMessage.FindMessage Method (Int32, String, String)

Search for a message in the current message store.

[Visual Basic]

```
Overloads Public Function FindMessage( _
   ByVal messageIndex As Integer, _
   ByVal headerField As String, _
   ByVal headerValue As String _
) As Integer
```

[C#]

```
public int FindMessage(
   int messageIndex,
   string headerField,
   string headerValue
);
```

Parameters

messageIndex
An integer value which specifies the message number that should be used when starting the search.

headerField
A string which specifies the name of the header field that should be searched. The header field name is not case sensitive.

headerValue
A string which specifies the header value that should be searched for. The search options can be used to specify if the search is case-sensitive, and whether the search should return partial matches to the string.

Return Value

An integer value which specifies the message index for the message that matched the search criteria. If no matching message was found, this method will return zero.

Remarks

The **FindMessage** method is used to search the message store for a message which matches a specific header field value. For example, it can be used to find every message which is addressed to a specific recipient or has a subject which matches a particular string value.

This implementation searches for a complete match to the specified header value, and the comparison is not case-sensitive.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.FindMessage Overload List

MailMessage.FindMessage Method (String, String, MimeSearch)

Search for a message in the current message store.

[Visual Basic]

```
Overloads Public Function FindMessage( _
   ByVal headerField As String, _
   ByVal headerValue As String, _
   ByVal searchOptions As MimeSearch _
) As Integer
```

[C#]

```
public int FindMessage(
   string headerField,
   string headerValue,
   MimeSearch searchOptions
);
```

Parameters

headerField

A string which specifies the name of the header field that should be searched. The header field name is not case sensitive.

headerValue

A string which specifies the header value that should be searched for. The search options can be used to specify if the search is case-sensitive, and whether the search should return partial matches to the string.

searchOptions

A MimeSearch enumeration which specifies the options to be used when searching for a message.

Return Value

An integer value which specifies the message index for the message that matched the search criteria. If no matching message was found, this method will return zero.

Remarks

The **FindMessage** method is used to search the message store for a message which matches a specific header field value. For example, it can be used to find every message which is addressed to a specific recipient or has a subject which matches a particular string value.

When searching for specific email addresses, it is recommended that you use the **searchPartialMatch** option because of the different ways that addresses can be annotated.

This implementation of the method always begins the search with the first message in the message store. To iterate through multiple matches to a header value, you must specify a message index.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.FindMessage Overload List

MailMessage.FindMessage Method (String, String)

Search for a message in the current message store.

[Visual Basic]

```
Overloads Public Function FindMessage( _
   ByVal headerField As String, _
   ByVal headerValue As String _
) As Integer
```

[C#]

```
public int FindMessage(
   string headerField,
   string headerValue
);
```

Parameters

headerField
: A string which specifies the name of the header field that should be searched. The header field name is not case sensitive.

headerValue
: A string which specifies the header value that should be searched for. The search options can be used to specify if the search is case-sensitive, and whether the search should return partial matches to the string.

Return Value

An integer value which specifies the message index for the message that matched the search criteria. If no matching message was found, this method will return zero.

Remarks

The **FindMessage** method is used to search the message store for a message which matches a specific header field value. For example, it can be used to find every message which is addressed to a specific recipient or has a subject which matches a particular string value.

This implementation searches for a complete match to the specified header value, and the comparison is not case-sensitive. The search always begins with the first message in the message store. To iterate through multiple matches to a header value, you must specify a message index.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.FindMessage Overload List

MailMessage.GetFirstHeader Method

Return the first header in the current message part.

[Visual Basic]

```
Public Function GetFirstHeader( _
   ByRef headerName As String, _
   ByRef headerValue As String _
) As Boolean
```

[C#]

```
public bool GetFirstHeader(
   ref string headerName,
   ref string headerValue
);
```

Parameters

headerName

A string passed by reference which will contain the name of the first header field when the method returns.

headerValue

A string passed by reference which will contain the value of the first header field when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFirstHeader** method allows an application to enumerate all of the headers in the current message part. If the current message part does not contain any header fields, this method will return **false**.

The current message part is returned by the **Part** property.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.GetHeader Method

Return the value of a header field in the specified message part.

Overload List

Return the value of a header field in the specified message part.

public bool GetHeader(int,string,ref string);

Return the value of a header field in the current message part.

public bool GetHeader(string,ref string);

See Also

MailMessage Class | SocketTools Namespace

MailMessage.GetHeader Method (Int32, String, String)

Return the value of a header field in the specified message part.

[Visual Basic]

```
Overloads Public Function GetHeader( _
   ByVal messagePart As Integer, _
   ByVal headerName As String, _
   ByRef headerValue As String _
) As Boolean
```

[C#]

```
public bool GetHeader(
   int messagePart,
   string headerName,
   ref string headerValue
);
```

Parameters

messagePart
An integer which specifies the message part.

headerName
A string which specifies the name of the header field.

headerValue
A string passed by reference which will contain the value of the header field when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeader** method is used to retrieve the value for a specific header in the specified message part. If the header field exists, the method will return **true** and the ***headerValue*** argument will contain the header value. If the header does not exist, the method will return **false**.

If there are multiple headers with the same name, the first value will be returned. To enumerate all of the headers in a message, including duplicate header fields, use the **GetFirstHeader** and **GetNextHeader** methods.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.GetHeader Overload List

MailMessage.GetHeader Method (String, String)

Return the value of a header field in the current message part.

[Visual Basic]

```
Overloads Public Function GetHeader( _
   ByVal headerName As String, _
   ByRef headerValue As String _
) As Boolean
```

[C#]

```
public bool GetHeader(
   string headerName,
   ref string headerValue
);
```

Parameters

headerName
A string which specifies the name of the header field.

headerValue
A string passed by reference which will contain the value of the header field when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeader** method is used to retrieve the value for a specific header in the current message part. If the header field exists, the method will return **true** and the ***headerValue*** argument will contain the header value. If the header does not exist, the method will return **false**.

If there are multiple headers with the same name, the first value will be returned. To enumerate all of the headers in a message, including duplicate header fields, use the **GetFirstHeader** and **GetNextHeader** methods.

The current message part is returned by the **Part** property.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.GetHeader Overload List

MailMessage.GetNextHeader Method

Return the next header in the current message part.

[Visual Basic]

```
Public Function GetNextHeader( _
   ByRef headerName As String, _
   ByRef headerValue As String _
) As Boolean
```

[C#]

```
public bool GetNextHeader(
   ref string headerName,
   ref string headerValue
);
```

Parameters

headerName
A string passed by reference which will contain the name of the first header field when the method returns.

headerValue
A string passed by reference which will contain the value of the first header field when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetNextHeader** method allows an application to enumerate all of the headers in the current message part. If the current message part does not contain any header fields, this method will return **false**.

The current message part is returned by the **Part** property.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.ImportMessage Method

Replace the current message with the contents of a file.

[Visual Basic]

```
Public Function ImportMessage( _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool ImportMessage(
   string fileName
);
```

Parameters

fileName

A string which specifies the name of the text file to import.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Initialize Method

Initialize an instance of the MailMessage class.

Overload List

Initialize an instance of the MailMessage class.

public bool Initialize();

Initialize an instance of the MailMessage class.

public bool Initialize(string);

See Also

MailMessage Class | SocketTools Namespace | Uninitialize Method

MailMessage.Initialize Method ()

Initialize an instance of the MailMessage class.

[Visual Basic]
```
Overloads Public Function Initialize() As Boolean
```

[C#]
```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the MailMessage class, allocating resources for the current thread. Typically it is not necessary to explicitly call this method because the instance of the class is initialized by the class constructor. However, if the **Uninitialize** method is called, the class must be re-initialized before any other methods are called.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.Initialize Overload List | Uninitialize Method

MailMessage.Initialize Method (String)

Initialize an instance of the MailMessage class.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the MailMessage class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the MailMessage class.

Example

The following example shows how to use the Initialize method to initialize an instance of the class. This example assumes that the license key string has been defined in code.

```
SocketTools.MailMessage mimeClient = new SocketTools.MailMessage();

if (mimeClient.Initialize(strLicenseKey) == false)
{
    MessageBox.Show(mimeClient.LastErrorString, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    return;
}
```

```
Dim mimeClient As New SocketTools.MailMessage

If mimeClient.Initialize(strLicenseKey) = False Then
    MsgBox(mimeClient.LastErrorString, vbIconExclamation)
    Exit Sub
End If
```

See Also

MailMessage Class | SocketTools Namespace | MailMessage.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

MailMessage.OpenStore Method

Open the specified message storage file for read-only access.

Overload List

Open the specified message storage file for read-only access.

public bool OpenStore(string);

Open the specified message storage file.

public bool OpenStore(string,MimeStorage);

See Also

MailMessage Class | SocketTools Namespace

MailMessage.OpenStore Method (String, MimeStorage)

Open the specified message storage file.

[Visual Basic]

```
Overloads Public Function OpenStore( _
   ByVal fileName As String, _
   ByVal openMode As MimeStorage _
) As Boolean
```

[C#]

```
public bool OpenStore(
   string fileName,
   MimeStorage openMode
);
```

Parameters

fileName
A string which specifies the name of the storage file.

openMode
A MimeStorage enumeration which specifies how the storage file should be opened.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **OpenStore** method opens a message storage file which contains one or more messages. If the storage file is opened for read access, the application can search the message store and extract messages but it cannot add or delete messages. To add new messages or delete existing messages from the store, it must be opened with write access.

The message store is designed to be a simple, effective way to store multiple messages together in a single file. When the message store is opened, the contents are indexed in memory. Although there is no specific limit to the number of messages that can be stored, there must be sufficient memory available to build an index of each message and its headers. If the application must store and manage a very large number of messages, it is recommended that you use a database rather than a flat-file message store.

Message Store Format

Each message is prefixed by a control sequence of five ASCII 01 characters followed by an ASCII 10 and ASCII 13 character. The messages themselves are stored unmodified in their original text format. The length of each message is calculated based on the location of the control sequence that delimits each message, and explicit message lengths are not stored in the file. This means that it is safe to manually change the message contents, as long as the message delimiters are preserved.

If the message store is compressed, the contents of the storage file are expanded when the file is opened and then re-compressed when the storage file is closed. Using the **MimeStorage.storeCompress** option reduces the size of the storage file and prevents the contents of the message store from being read using a text file editor. However, enabling compression will increase the amount of memory allocated by the class and can increase the amount of time that it takes to open and close the storage file.

The class also has a backwards compatibility mode where it will recognize storage files that use the UNIX mbox format. While this format is supported for accessing existing files, it is not recommended that you use it when creating new message stores or adding messages to an existing store. There are a number of different variants on the mbox format that have been used by different Mail Transfer Agents (MTAs) on the UNIX platform. For example, the mboxrd variant looks identical to the mboxcl2 variant, and they are programmatically indistinguishable from one another, but they are not compatible. For this reason, the use of the mbox format is strongly discouraged.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.OpenStore Overload List

MailMessage.OpenStore Method (String)

Open the specified message storage file for read-only access.

[Visual Basic]

```
Overloads Public Function OpenStore( _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool OpenStore(
   string fileName
);
```

Parameters

fileName

A string which specifies the name of the storage file.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **OpenStore** method opens a message storage file for read-only access. The application can search the message store and extract messages but it cannot add or delete messages. To add new messages or delete existing messages from the store, it must be opened with write access.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.OpenStore Overload List

MailMessage.ParseAddress Method

Parse an Internet email address.

Overload List

Parse an Internet email address.

public bool ParseAddress(string,string,ref string);

Parse an Internet email address.

public bool ParseAddress(string,ref string);

See Also

MailMessage Class | SocketTools Namespace

MailMessage.ParseAddress Method (String, String, String)

Parse an Internet email address.

[Visual Basic]

```
Overloads Public Function ParseAddress( _
   ByVal mailAddress As String, _
   ByVal mailDomain As String, _
   ByRef parsedAddress As String _
) As Boolean
```

[C#]

```
public bool ParseAddress(
   string mailAddress,
   string mailDomain,
   ref string parsedAddress
);
```

Parameters

mailAddress
A string which specifies the email address to be parsed.

mailDomain
A string which specifies a default domain for the address if no domain name is specified in the *mailAddress* parameter.

parsedAddress
A string passed by reference which will contain the parsed email address.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ParseAddress** method is useful for parsing the email addresses that may be specified in various header fields in the message. In many cases, the addresses have additional comment characters which are not part of the address itself. For example, one common format is "User Name" <user@domain.com>. In this case, the email address is enclosed in angle brackets and the name outside of the brackets is considered to be a comment which is not part of the address itself.

Another common format is user@domain.com (User Name). In this case, there is the address followed by a comment which is enclosed in parenthesis. The **ParseAddress** method recognizes both formats, and when passed either string, would return the address user@domain.com.

If there was no domain specified in the address, that is just a user name was specified, then the value the *mailDomain* parameter is added to the address.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.ParseAddress Overload List

MailMessage.ParseAddress Method (String, String)

Parse an Internet email address.

```
[Visual Basic]
Overloads Public Function ParseAddress( _
   ByVal mailAddress As String, _
   ByRef parsedAddress As String _
) As Boolean
```

```
[C#]
public bool ParseAddress(
   string mailAddress,
   ref string parsedAddress
);
```

Parameters

mailAddress
A string which specifies the email address to be parsed.

parsedAddress
A string passed by reference which will contain the parsed email address.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ParseAddress** method is useful for parsing the email addresses that may be specified in various header fields in the message. In many cases, the addresses have additional comment characters which are not part of the address itself. For example, one common format is "User Name" <user@domain.com>. In this case, the email address is enclosed in angle brackets and the name outside of the brackets is considered to be a comment which is not part of the address itself.

Another common format is user@domain.com (User Name). In this case, there is the address followed by a comment which is enclosed in parenthesis. The **ParseAddress** method recognizes both formats, and when passed either string, would return the address user@domain.com.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.ParseAddress Overload List

MailMessage.ParseMessage Method

Parse the specified string, adding the contents to the current message.

[Visual Basic]

```
Public Function ParseMessage( _
   ByVal messageText As String _
) As Boolean
```

[C#]

```
public bool ParseMessage(
   string messageText
);
```

Parameters

messageText
A string which contains the message text to be parsed.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ParseMessage** method parses a string which contains message data, adding it to the current message. This method is useful when the application needs to parse an arbitrary block of text and add it to the current message. If the string contains header fields, the values will be added to the message header. Once the end of the header block is detected, all subsequent text is added to the body of the message.

Note that unlike the **ImportMessage** method, the **ParseMessage** method does not clear the contents of the current message and may be called multiple times. Use the **ClearMessage** method to clear the current message before calling **ParseMessage** if necessary.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.PurgeStore Method

Purge all deleted messages from the current message store.

[Visual Basic]

```
Public Function PurgeStore() As Boolean
```

[C#]

```
public bool PurgeStore();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PurgeStore** method purges all deleted messages from the message store. If the storage file has been opened in read-only mode or there are no messages marked for deletion, this method will take no action.

When the **CloseStore** method is called, the storage file will automatically be purged. To prevent deleted messages from being removed from the message store, use the **Reset** method.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. All properties will be reset to their default values.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.ReadStore Method

Retrieve a message from the message store, replacing the current message.

Overload List

Retrieve a message from the message store, replacing the current message.

public bool ReadStore();

Retrieve a message from the message store, replacing the current message.

public bool ReadStore(int);

See Also

MailMessage Class | SocketTools Namespace

MailMessage.ReadStore Method (Int32)

Retrieve a message from the message store, replacing the current message.

[Visual Basic]

```
Overloads Public Function ReadStore( _
   ByVal messageIndex As Integer _
) As Boolean
```

[C#]

```
public bool ReadStore(
   int messageIndex
);
```

Parameters

messageIndex

An integer value which specifies the message that will be removed from the message store.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ReadStore** method reads the specified message from the message store, and the contents of that message will replace the current message.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.ReadStore Overload List

MailMessage.ReadStore Method ()

Retrieve a message from the message store, replacing the current message.

[Visual Basic]

```
Overloads Public Function ReadStore() As Boolean
```

[C#]

```
public bool ReadStore();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ReadStore** method reads a message from the message store, and the contents of that message will replace the current message. The **StoreIndex** property specifies the current message index into the storage file.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.ReadStore Overload List

MailMessage.ReplaceMessage Method

Replace the current message in the message store.

Overload List

Replace the current message in the message store.

public bool ReplaceMessage();

Replace the specified message in the current message store.

public bool ReplaceMessage(int);

See Also

MailMessage Class | SocketTools Namespace

MailMessage.ReplaceMessage Method (Int32)

Replace the specified message in the current message store.

[Visual Basic]

```
Overloads Public Function ReplaceMessage( _
   ByVal messageIndex As Integer _
) As Boolean
```

[C#]

```
public bool ReplaceMessage(
   int messageIndex
);
```

Parameters

messageIndex
An integer value which specifies the message that will be replaced in the message store.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ReplaceMessage** method replaces the specified message with a new message. The message number may be a message that has been previously marked for deletion. It is important to note that the change will not be reflected in the physical storage file until it has been closed. This method will update the current message index in the storage file.

The message store must be opened with write access. This method will fail if you attempt to replace a message from a storage file that has been opened for read-only access. If the application needs to replace messages in the message store, it is recommended that the file be opened for exclusive access using the **storeLock** option when calling the **OpenStore** method.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.ReplaceMessage Overload List

MailMessage.ReplaceMessage Method ()

Replace the current message in the message store.

[Visual Basic]
```
Overloads Public Function ReplaceMessage() As Boolean
```

[C#]
```
public bool ReplaceMessage();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ReplaceMessage** method replaces the current message in the message store. The current message index into the storage file is specified by the **StoreIndex** property and this may be a message that has been previously marked for deletion. It is important to note that the change will not be reflected in the physical storage file until it has been closed. This method will update the current message index in the storage file.

The message store must be opened with write access. This method will fail if you attempt to replace a message from a storage file that has been opened for read-only access. If the application needs to replace messages in the message store, it is recommended that the file be opened for exclusive access using the **storeLock** option when calling the **OpenStore** method.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.ReplaceMessage Overload List

MailMessage.SetHeader Method

Set the value for a header in the specified message part.

Overload List

Set the value for a header in the specified message part.

public bool SetHeader(int,string,string);

Set the value for a header in the current message part.

public bool SetHeader(string,string);

See Also

MailMessage Class | SocketTools Namespace

MailMessage.SetHeader Method (Int32, String, String)

Set the value for a header in the specified message part.

[Visual Basic]

```
Overloads Public Function SetHeader( _
   ByVal messagePart As Integer, _
   ByVal headerName As String, _
   ByVal headerValue As String _
) As Boolean
```

[C#]

```
public bool SetHeader(
   int messagePart,
   string headerName,
   string headerValue
);
```

Parameters

messagePart
: An integer which specifies the message part.

headerName
: A string which specifies the name of the header field.

headerValue
: A string which specifies the value of the header field. If an empty string is specified, the header field is removed from the header block in the specified message part.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.SetHeader Overload List

MailMessage.SetHeader Method (String, String)

Set the value for a header in the current message part.

[Visual Basic]

```
Overloads Public Function SetHeader( _
   ByVal headerName As String, _
   ByVal headerValue As String _
) As Boolean
```

[C#]

```
public bool SetHeader(
   string headerName,
   string headerValue
);
```

Parameters

headerName

A string which specifies the name of the header field.

headerValue

A string which specifies the value of the header field. If an empty string is specified, the header field is removed from the header block in the current message part.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The current message part is returned by the **Part** property.

See Also

MailMessage Class | SocketTools Namespace | MailMessage.SetHeader Overload List

MailMessage.Uninitialize Method

Uninitialize the class library and release any resources allocated for the current thread.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method releases resources allocated for the current process. After this method has been called, no further operations may be performed until the class instance has been re-initialized.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the class is no longer needed.

See Also

MailMessage Class | SocketTools Namespace | Initialize Method

MailMessage.WriteStore Method

Store the current message in the message store.

[Visual Basic]

```
Public Function WriteStore() As Boolean
```

[C#]

```
public bool WriteStore();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **WriteStore** method will always append the current message to the storage file. If you want to replace a message in the message store, you should use the **ReplaceMessage** method.

This method will update the value of the **StoreIndex** property to specify the message number for the new message that has been added to the storage file.

See Also

MailMessage Class | SocketTools Namespace | ReplaceMessage Method

MailMessage Events

The events of the **MailMessage** class are listed below. For a complete list of **MailMessage** class members, see the MailMessage Members topic.

Public Instance Events

⚡ OnError	Occurs when an client operation fails.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.OnError Event

Occurs when an client operation fails.

[Visual Basic]

```
Public Event OnError As OnErrorEventHandler
```

[C#]

```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type MailMessage.ErrorEventArgs containing data related to this event. The following **MailMessage.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a client operation fails.

See Also

MailMessage Class | SocketTools Namespace

MailMessage.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see MailMessage.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.MailMessage.ErrorEventArgs

[Visual Basic]

```
Public Class MailMessage.ErrorEventArgs
    Inherits EventArgs
```

[C#]

```
public class MailMessage.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.MailMessage (in SocketTools.MailMessage.dll)

See Also

MailMessage.ErrorEventArgs Members | SocketTools Namespace

MailMessage.ErrorEventArgs Members

MailMessage.ErrorEventArgs overview

Public Instance Constructors

MailMessage.ErrorEventArgs Constructor	Initializes a new instance of the MailMessage.ErrorEventArgs class.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

MailMessage.ErrorEventArgs Class | SocketTools Namespace

MailMessage.ErrorEventArgs Constructor

Initializes a new instance of the MailMessage.ErrorEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public MailMessage.ErrorEventArgs();
```

See Also

MailMessage.ErrorEventArgs Class | SocketTools Namespace

MailMessage.ErrorEventArgs Properties

The properties of the **MailMessage.ErrorEventArgs** class are listed below. For a complete list of **MailMessage.ErrorEventArgs** class members, see the MailMessage.ErrorEventArgs Members topic.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

MailMessage.ErrorEventArgs Class | SocketTools Namespace

MailMessage.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

MailMessage.ErrorEventArgs Class | SocketTools Namespace | Error Property

MailMessage.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public MailMessage.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

MailMessage.ErrorEventArgs Class | SocketTools Namespace | Description Property

MailMessage.ErrorCode Enumeration

Specifies the error codes returned by the MailMessage class.

[Visual Basic]

```
Public Enum MailMessage.ErrorCode
```

[C#]

```
public enum MailMessage.ErrorCode
```

Remarks

The MailMessage class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.
errorProxyAuthenticationRequired	Proxy authentication required.

errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.
errorUnsupportedMediaType	Request specified an unsupported media type.

errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.
errorNoMoreMailboxes	No more mailboxes exist on this server.

errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.
errorInvalidThreadHandle	The specified thread handle is no longer valid.

errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been defined.

errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or unrecognized.

errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.

Requirements

Namespace: SocketTools

Assembly: SocketTools.MailMessage (in SocketTools.MailMessage.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

MailMessage.MimeAttachment Enumeration

Specifies the file attachment options supported by the MailMessage class.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum MailMessage.MimeAttachment
```

[C#]

```
[Flags]
public enum MailMessage.MimeAttachment
```

Remarks

The **attachAlternative** and **attachInline** values can be combined with one of the attachment options using a bitwise OR operator.

Members

Member Name	Description	Value
attachDefault	The file attachment encoding is based on the file content type. Text files are not encoded, and binary files are encoded using the standard base64 encoding algorithm. This is the default option for file attachments.	0
attachBase64	The file attachment is always encoded using the standard base64 algorithm, even if the attached file is a plain text file.	1
attachUucode	The file attachment is always encoded using the uuencode algorithm, even if the attached file is a plain text file.	2
attachQuoted	The file attachment is always encoded using the quoted-printable algorithm, even if the attached file is a plain text file.	3
attachAlternative	The attached data is an alternative format for the contents of the message. This can only be used with textual data.	65536
attachInline	The attached data will be displayed inline with the contents of the message. This is typically used with images that are to be displayed along with the message text.	131072

Requirements

Namespace: SocketTools

Assembly: SocketTools.MailMessage (in SocketTools.MailMessage.dll)

See Also

SocketTools Namespace

MailMessage.MimeCharacterSet Enumeration

Specifies the character sets recognized by the MailMessage class.

[Visual Basic]

```
Public Enum MailMessage.MimeCharacterSet
```

[C#]

```
public enum MailMessage.MimeCharacterSet
```

Members

Member Name	Description
charsetUnknown	The character set is unknown.
charsetDefault	The default character set. This is the same as specifying the character set **charsetUTF8**.
charsetUSASCII	A character set using US-ASCII which defines 7-bit printable characters with values ranging from 20h to 7Eh. An application that uses this character set has the broadest compatibility with most mail servers (MTAs) because it does not require the server to handle 8-bit characters correctly when the message is delivered. This is the most commonly used character set for plain text email messages in the English language and is the default character set used by the class.
charsetISO8859_1	An 8-bit character set for most western European languages such as English, French, Spanish and German. This character set is also commonly referred to as Latin1. The Windows code page for this character set is 28591, however Windows code page 1252 (Windows-1252) is typically used to represent this character set in most applications.
charsetISO8859_2	An 8-bit character set for most central and eastern European languages such as Czech, Hungarian, Polish and Romanian. This character set is also commonly referred to as Latin2. This character set is similar to Windows code page 1250, however the characters are arranged differently.
charsetISO8859_3	A character set for southern European languages such as Maltese and Esperanto. This character set was also used with the Turkish language, but it was superseded by ISO 8859-9 which is the preferred character set for Turkish. This character set is not widely used in mail messages and it is recommended that you use UTF-8 instead.
charsetISO8859_4	A character set for northern European languages such as Latvian, Lithuanian and Greenlandic. This character set is not widely used in mail messages

	and it is recommended that you use UTF-8 instead.
charsetISO8859_5	An 8-bit character set for Cyrillic languages such as Russian, Bulgarian and Serbian. The Windows code page for this character set is 28595. This character set is not widely used and it is recommended that you use UTF-8 instead.
charsetISO8859_6	An 8-bit character set for Arabic languages. Note that the application is responsible for displaying text that uses this character set. In particular, any display engine needs to be able to handle the reverse writing direction and analyze the context of the message to correctly combine the glyphs. This character set is not widely used and it is recommended that you use UTF-8 instead.
charsetISO8859_7	An 8-bit character set for the Greek language. This character set is also commonly referred to as Latin/Greek. The Windows code page for this character set is 28597.
charsetISO8859_8	An 8-bit character set for the Hebrew language. Note that similar to Arabic, Hebrew uses a reverse writing direction. An application which displays this character should be capable of processing bi-directional text where a single message may include both right-to-left and left-to-right languages, such as Hebrew and English. The Windows code page for this character set is 28598.
charsetISO8859_9	An 8-bit character set for the Turkish language. This character set is also commonly referred to as Latin5. The Windows code page for this character set is 28599.
charsetISO8859_10	A character set for the Danish, Icelandic, Norwegian and Swedish languages. This character set is also commonly referred to as Latin-6 and is similar to ISO 8859-4.
charsetISO8859_13	A character set for Baltic languages. This character set is also commonly referred to as Latin-7. This character set is similar to ISO 8859-4, except it adds certain Polish characters and does not support Nordic languages.
charsetISO8859_14	A character set for Gaelic languages such as Irish, Manx and Scottish Gaelic. This character set is also commonly referred to as Latin-8. This character set replaced ISO 8859-12 which was never fully implemented.
charsetISO8859_15	A character set for western European languages. This character set is also commonly referred to as Latin-9 and is nearly identical to ISO8859-1 except

	that it replaces lesser-used symbols with the Euro sign and some letters.
charsetISO2022_JP	A multi-byte character encoding for Japanese that is widely used with mail messages. This is a 7-bit encoding where all characters start with ASCII and uses escape sequences to switch to the double-byte character sets.
charsetISO2022_KR	A multi-byte character encoding for Korean which encodes both ASCII and Korean double-byte characters. This is a 7-bit encoding which uses the shift in and shift out control characters to switch to the double-byte character set.
charsetISO2022_CN	A multi-byte character encoding for Simplified Chinese which encodes both ASCII and Chinese double-byte characters. This is a 7-bit encoding which uses the shift in and shift out control characters to switch to the double-byte character set.
charsetKOI8R	A character set for Russian using the Cyrillic alphabet. This character set also covers the Bulgarian language. Most mail messages in the Russian language use this character set or UTF-8 instead of ISO 8859-5, which was never widely adopted.
charsetKOI8U	A character set for Ukrainian using the Cyrillic alphabet. This character set is similar to the KOI8-R character set, but replaces certain symbols with Ukrainian letters. Most mail messages in the Ukrainian language use this character set or UTF-8 instead of ISO 8859-5, which was never widely adopted.
charsetGB2312	A multi-byte character encoding which can represent ASCII and simplified Chinese characters. It has been superseded by GB18030, however it remains widely used in China.
charsetGB18030	A Unicode transformation format which can represent all Unicode code points and supports both simplified and traditional Chinese characters. It is backwards compatible with GB2312 and supersedes that character set.
charsetBIG5	A multi-byte character set that supports both ASCII characters and traditional Chinese characters. It is widely used in Taiwan, Hong Kong and Macau. It is no longer commonly used in China, which has developed GB18030 as a standard encoding. Note that Microsoft's implementation of Big5 on Windows does not support all of the extensions and is missing certain

	code points.
charsetUTF7	A 7-bit Unicode Transformation Format that uses variable-length character encoding to represent Unicode text as a stream of ASCII characters that are safe to transport between mail servers that only support 7-bit printable characters. It is primarily used as an alternative to UTF-8 which requires that the mail server support 8-bit text or use quoted-printable encoding.
charsetUTF8	An 8-bit Unicode Transformation Format that uses multibyte character sequences to represent Unicode text. It is backwards compatible with the ASCII character set, however because it uses 8-bit text, it should be encoded using either quoted-printable or base64 encoding to ensure that mail servers that do not support 8-bit characters.
charsetUTF16	A 16-bit Unicode Transformation Format that uses two bytes to represent each Unicode character. Messages that use UTF-16 are commonly encoded using the base64 algorithm. It is recommended that most applications use the UTF-8 character set, which is capable of representing all Unicode characters.

Requirements

Namespace: SocketTools

Assembly: SocketTools.MailMessage (in SocketTools.MailMessage.dll)

See Also

SocketTools Namespace

MailMessage.MimeContent Enumeration

Specifies the content types supported by the MailMessage class.

[Visual Basic]
```
Public Enum MailMessage.MimeContent
```

[C#]
```
public enum MailMessage.MimeContent
```

Members

Member Name	Description
contentUnknown	The content type is unknown. This value may be returned if the message handle is invalid, or if the file extension is unknown and the file could not be opened for read access.
contentDefault	The default content type. This is the same as specifying the content type **contentText**.
contentApplication	The content is application specific. Examples of this type of file would be a Microsoft Word document or an executable program. This is also the default type for files which have an unrecognized file name extension and contain binary data.
contentAudio	The content is audio data in one of several standard formats. Examples of this type of file would be a Windows (.wav) file or MPEG3 (.mp3) file.
contentImage	The content is an image data in one of several standard formats. Examples of this type of file would be a GIF or JPEG image file.
contentMessage	The content is an email message encapsulated within the current message.
contentMultipart	The content is a multipart MIME email message which contains additional message parts. For example, an email message which contains both a text message and file attachment would be identified as a multipart message.
contentText	The content is textual data. This is also the default type for files which have an unrecognized file name extension and contain only printable text.
contentVideo	The content is video data in one of several standard formats. Examples of this type of file would be a Windows (.avi) or Quicktime (.mov) video file.
contentWideText	The content is Unicode text. This is also the default type for files which have an unrecognized file

	name extension. The content must be prefixed with a byte order mark (BOM) to be recognized as Unicode text.

Requirements

Namespace: SocketTools

Assembly: SocketTools.MailMessage (in SocketTools.MailMessage.dll)

See Also

SocketTools Namespace

MailMessage.MimeEncoding Enumeration

Specifies the encoding types supported by the MailMessage class.

[Visual Basic]
```
Public Enum MailMessage.MimeEncoding
```

[C#]
```
public enum MailMessage.MimeEncoding
```

Members

Member Name	Description
encodingUnknown	The encoding type is unknown.
encodingDefault	The encoding type is based on the content type. Textual data is not encoded, and binary data is encoded using the standard base64 encoding algorithm. This is the default option for file attachments.
encoding7Bit	The data contains printable UTF-8 and ANSI text characters. This is the default encoding type for most email messages.
encoding8Bit	The data contains printable ASCII characters as well as printable characters using UTF-8 encoding and the ISO8859 (Latin1) character set. In some cases, it may be necessary to use basse64 encoding if the mail server cannot accept 8-bit characters.
encodingBinary	The data contains unmodified 8-bit binary data. Messages which contain binary data should be encoded using the base64 algorithm.
encodingQuoted	The data is encoded using quoted-printable encoding. Printable ASCII characters are left as-is, with non-printable characters encoded as their hexadecimal value. Quoted-printable encoding is commonly used with HTML formatted email messages.
encodingBase64	The data is encoded using the standard base64 algorithm. This is the most common encoding method for binary data in an email message.
encodingUucode	The data is encoded using the uuencode algorithm. This encoding method is common for binary attachments to news articles, but is rarely used with email messages.

Requirements

Namespace: SocketTools

Assembly: SocketTools.MailMessage (in SocketTools.MailMessage.dll)

See Also

SocketTools Namespace

MailMessage.MimeExportOptions Enumeration

Specifies the export options that the MailMessage class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum MailMessage.MimeExportOptions
```

[C#]

```
[Flags]
public enum MailMessage.MimeExportOptions
```

Members

Member Name	Description	Value
exportDefault	The default export options. The headers for the message are written out in a specific consistent order, with custom headers written to the end of the header block regardless of the order in which they were set or imported from another message. If the message contains Bcc, Received, Return-Path, Status or X400-Received header fields, they will not be exported.	0
exportAllHeaders	All headers, including the Bcc, Received, Return-Path, Status and X400-Received header fields will be exported. Normally these headers are not exported because they are only used by the mail transport system. This option can be useful when exporting a message to be stored on the local system, but should not be used when exporting a message to be delivered to another user.	1
exportKeepOrder	The original order in which the message header fields were set or imported are preserved when the message is exported.	2
exportNoHeaders	When exporting a message, the main header block will not be included. This can be useful when creating a multipart message for services which expect MIME formatted data, such as HTTP POST requests. This option should never be used for email messages being submitted using SMTP.	4

Requirements

Namespace: SocketTools

Assembly: SocketTools.MailMessage (in SocketTools.MailMessage.dll)

See Also

SocketTools Namespace

MailMessage.MimeSearch Enumeration

Specifies the search options supported by the MailMessage class.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum MailMessage.MimeSearch
```

[C#]
```
[Flags]
public enum MailMessage.MimeSearch
```

Members

Member Name	Description	Value
searchDefault	Perform a complete match against the specified header value. The comparison is not case-sensitive. It is the default search option used if a search option is not specified.	0
searchCaseSensitive	The header value comparison will be case-sensitive. Note that this does not affect header field names. Matches for header names are always case-insensitive.	1
searchPartialMatch	Perform a partial match against the specified header value. It recommended that this option be used when searching for matches to email addresses.	2
searchDecodeHeaders	Decode any encoded message headers before comparing them to the specified value. This option can increase the amount of time required to search the message store and should only be used when necessary.	4

Requirements

Namespace: SocketTools

Assembly: SocketTools.MailMessage (in SocketTools.MailMessage.dll)

See Also

SocketTools Namespace | FindMessage Method (SocketTools.MailMessage)

MailMessage.MimeStorage Enumeration

Specifies the message store access options supported by the MailMessage class.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum MailMessage.MimeStorage
```

[C#]

```
[Flags]
public enum MailMessage.MimeStorage
```

Members

Member Name	Description	Value
storeRead	The message store will be opened for read access. The contents of the message store can be accessed, but cannot be modified by the process unless it has also been opened for writing.	0
storeWrite	The message store will be opened for writing. This mode also implies read access and must be specified if the application needs to modify the contents of the message store.	1
storeCreate	The message store will be created if the storage file does not exist. If the file exists, it will be truncated. This mode implies read and write access.	2
storeLock	The message store will be opened so that it may only be accessed and modified by the current process.	4
storeCompress	The contents of the message store are compressed. This option is automatically enabled if a compressed message store is opened for reading or writing.	4096
storeMailbox	The message store should use the UNIX mbox format when reading and storing messages. This option is provided for backwards compatibility and is not recommended for general use.	8192

Requirements

Namespace: SocketTools

Assembly: SocketTools.MailMessage (in SocketTools.MailMessage.dll)

See Also

SocketTools Namespace | OpenStore Method (SocketTools.MailMessage)

MailMessage.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub MailMessage.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void MailMessage.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ErrorEventArgs that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.MailMessage (in SocketTools.MailMessage.dll)

See Also

SocketTools Namespace

MailMessage.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see MailMessage.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.MailMessage.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class MailMessage.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class MailMessage.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the MailMessage class.

Example

```
<Assembly: SocketTools.MailMessage.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.MailMessage.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.MailMessage (in SocketTools.MailMessage.dll)

See Also

MailMessage.RuntimeLicenseAttribute Members | SocketTools Namespace

MailMessage.RuntimeLicenseAttribute Members

MailMessage.RuntimeLicenseAttribute overview

Public Instance Constructors

MailMessage.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

MailMessage.RuntimeLicenseAttribute Class | SocketTools Namespace

MailMessage.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public MailMessage.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the MailMessage class.

See Also

MailMessage.RuntimeLicenseAttribute Class | SocketTools Namespace

MailMessage.RuntimeLicenseAttribute Properties

The properties of the **MailMessage.RuntimeLicenseAttribute** class are listed below. For a complete list of **MailMessage.RuntimeLicenseAttribute** class members, see the MailMessage.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

MailMessage.RuntimeLicenseAttribute Class | SocketTools Namespace

MailMessage.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

MailMessage.RuntimeLicenseAttribute Class | SocketTools Namespace

MailMessageException Class

The exception that is thrown when a client error occurs.

For a list of all members of this type, see MailMessageException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.MailMessageException

[Visual Basic]

```
Public Class MailMessageException
    Inherits ApplicationException
```

[C#]

```
public class MailMessageException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A MailMessageException is thrown by the MailMessage class when an error occurs.

The default constructor for the MailMessageException class sets the **ErrorCode** property to the last client error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.MailMessage (in SocketTools.MailMessage.dll)

See Also

MailMessageException Members | SocketTools Namespace

MailMessageException Members

MailMessageException overview

Public Instance Constructors

MailMessageException	Overloaded. Initializes a new instance of the MailMessageException class.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

MailMessageException Class | SocketTools Namespace

MailMessageException Constructor

Initializes a new instance of the MailMessageException class with the last network error code.

Overload List

Initializes a new instance of the MailMessageException class with the last network error code.

public MailMessageException();

Initializes a new instance of the MailMessageException class with a specified error number.

public MailMessageException(int);

Initializes a new instance of the MailMessageException class with a specified error message.

public MailMessageException(string);

Initializes a new instance of the MailMessageException class with a specified error message and a reference to the inner exception that is the cause of this exception.

public MailMessageException(string,Exception);

See Also

MailMessageException Class | SocketTools Namespace

MailMessageException Constructor ()

Initializes a new instance of the MailMessageException class with the last network error code.

[Visual Basic]

```
Overloads Public Sub New()
```

[C#]

```
public MailMessageException();
```

Remarks

The ctor constructor sets the **ErrorCode** property to the last client error that occurred. For more information about the errors that may occur, refer to the MailMessage.ErrorCode enumeration.

See Also

MailMessageException Class | SocketTools Namespace | MailMessageException Constructor Overload List

MailMessageException Constructor (String)

Initializes a new instance of the MailMessageException class with a specified error message.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]

```
public MailMessageException(
   string message
);
```

Parameters

message
The error message that explains the reason for the exception.

Remarks

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

MailMessageException Class | SocketTools Namespace | MailMessageException Constructor Overload List

MailMessageException Constructor (String, Exception)

Initializes a new instance of the MailMessageException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal innerException As Exception _
)
```

```
[C#]
public MailMessageException(
   string message,
   Exception innerException
);
```

Parameters

message
The error message that explains the reason for the exception.

innerException
The exception that is the cause of the current exception. If the ***innerException*** parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

MailMessageException Class | SocketTools Namespace | MailMessageException Constructor Overload List

MailMessageException Constructor (Int32)

Initializes a new instance of the MailMessageException class with a specified error number.

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

```
[C#]
public MailMessageException(
   int code
);
```

Parameters

code

An integer value which specifies an error code.

Remarks

This constructor sets the **ErrorCode** property to the specified error code. For more information about the errors that may occur, refer to the MailMessage.ErrorCode enumeration.

See Also

MailMessageException Class | SocketTools Namespace | MailMessageException Constructor Overload List

MailMessageException Properties

The properties of the **MailMessageException** class are listed below. For a complete list of **MailMessageException** class members, see the MailMessageException Members topic.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

MailMessageException Class | SocketTools Namespace

MailMessageException.ErrorCode Property

Gets a value which specifies the error that caused the exception.

[Visual Basic]
```
Public ReadOnly Property ErrorCode As ErrorCode
```

[C#]
```
public MailMessage.ErrorCode ErrorCode {get;}
```

Property Value

Returns a MailMessage.ErrorCode enumeration value which specifies the error code.

Remarks

The **ErrorCode** property returns the error code that specifies the cause of the exception.

The default constructor for the MailMessageException class sets the error code to the last network error that occurred. For more information about the errors that may occur, refer to the MailMessage.ErrorCode enumeration.

See Also

MailMessageException Class | SocketTools Namespace

MailMessageException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]

```
Overrides Public ReadOnly Property Message As String
```

[C#]

```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

MailMessageException Class | SocketTools Namespace

MailMessageException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception. This value is the same as the values returned by the Windows Sockets API. For more information about socket error codes, see the Windows Socket Version 2 API error code documentation in MSDN.

See Also

MailMessageException Class | SocketTools Namespace

MailMessageException Methods

The methods of the **MailMessageException** class are listed below. For a complete list of **MailMessageException** class members, see the MailMessageException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

MailMessageException Class | SocketTools Namespace

MailMessageException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]

```
Overrides Public Function ToString() As String
```

[C#]

```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

MailMessageException Class | SocketTools Namespace

NntpClient Class

Implements the Network News Transfer Protocol.

For a list of all members of this type, see NntpClient Members.

System.Object
  SocketTools.NntpClient

```
[Visual Basic]
Public Class NntpClient
    Implements IDisposable
```

```
[C#]
public class NntpClient : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The Network News Transfer Protocol (NNTP) is used with servers that provide news services. This is similar in functionality to bulletin boards or message boards, where topics are organized hierarchically into groups, called newsgroups. Users can browse and search for messages, called news articles, which have been posted by other users. On many servers, they can also post their own articles which can be read by others. The largest collection of public newsgroups available is called USENET, a world-wide distributed discussion system. In addition, there are a large number of smaller news servers. For example, Microsoft operates a news server which functions as a forum for technical questions and announcements.

The NntpClient class provides a comprehensive interface for accessing newsgroups, retrieving articles and posting new articles. In combination with the MailMessage class to process the news articles, this class can be used to integrate newsgroup access with an existing email application, or you can implement your own full-featured newsgroup client.

This class supports secure connections using the standard SSL and TLS protocols.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NntpClient (in SocketTools.NntpClient.dll)

See Also

NntpClient Members | SocketTools Namespace

NntpClient Members

NntpClient overview

Public Static (Shared) Fields

nntpPortDefault	A constant value which specifies the default port number.
nntpPortSecure	A constant value which specifies the default port number for a secure connection.
nntpTimeout	A constant value which specifies the default timeout period.

Public Instance Constructors

NntpClient Constructor	Initializes a new instance of the NntpClient class.

Public Instance Properties

Article	Gets and sets the current article number in the selected newsgroup.
ArticleCount	Gets the number of available articles in the current newsgroup.
AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
Blocking	Gets and sets a value which indicates if the client is in blocking mode.
CertificateExpires	Get a value that specifies the date that the security certificate expires.
CertificateIssued	Get a value that specifies the date that the security certificate was issued.
CertificateIssuer	Get a value that provides information about the organization that issued the certificate.
CertificateName	Gets and sets a value that specifies the name of the client certificate.
CertificatePassword	Gets and sets the password associated with the client certificate.
CertificateStatus	Gets a value which indicates the status of the security certificate returned by the remote host.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateSubject	Gets a value that provides information about the organization that the server certificate was issued to.
CertificateUser	Gets and sets the user that owns the client certificate.

CipherStrength	Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.
FirstArticle	Gets the first available article in the selected newsgroup.
GroupName	Gets and sets the name of the currently selected newsgroup.
GroupTitle	Gets the title associated with the currently selected newsgroup.
Handle	Gets a value that specifies the client handle allocated for the current session.
HashStrength	Gets a value which specifies the length of the message digest that was selected for a secure connection.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsConnected	Gets a value which indicates if a connection to the remote host has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsReadable	Gets a value which indicates if there is data available to be read from the socket connection to the server.
IsWritable	Gets a value which indicates if data can be written to the client without blocking.
LastArticle	Gets the last available article in the selected newsgroup.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets the local Internet address that the client is bound to.
Localize	Gets a value that specifies if the date and time are localized.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
MessageId	Gets the unique message identifier for the current

	news article.
Options	Gets and sets a value which specifies one or more client options.
Password	Gets and sets the password used to authenticate the client session.
RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.
ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.
Secure	Gets and sets a value which specifies if a secure connection is established.
SecureCipher	Gets a value that specifies the encryption algorithm used for a secure connection.
SecureHash	Gets a value that specifies the message digest algorithm used for a secure connection.
SecureKeyExchange	Gets a value that specifies the key exchange algorithm used for a secure connection.
SecureProtocol	Gets and sets a value which specifies the protocol used for a secure connection.
Status	Gets a value which specifies the current status of the client.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
TimeZone	Gets and sets the current timezone offset in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
UserName	Gets and sets the username used to authenticate the client session.

Version	Gets a value which returns the current version of the NntpClient class library.

Public Instance Methods

AttachThread	Attach an instance of the class to the current thread
Authenticate	Overloaded. Authenticate the client session with a username and password.
Cancel	Cancel the current blocking client operation.
CloseArticle	Closes the current article that has been opened or created.
Command	Overloaded. Send a custom command to the server.
Connect	Overloaded. Establish a connection with a remote host.
CreateArticle	Creates a new article in the current newsgroup.
Disconnect	Terminate the connection with a remote host.
Dispose	Overloaded. Releases all resources used by NntpClient.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetArticle	Overloaded. Retrieve an article from the server.
GetFirstArticle	Overloaded. Return information about a selected range of articles in the current newsgroup.
GetFirstGroup	Overloaded. Return information about newsgroups created after the specified date.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetHeaders	Overloaded. Retrieves the headers for the specified article from the server.
GetNextArticle	Return information about the next article available in the current newsgroup.
GetNextGroup	Return information about the next available newsgroup.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the NntpClient class.
OpenArticle	Overloaded. Opens the specified article in the currently selected newsgroup.
PostArticle	Overloaded. Post a new article to the currently selected newsgroup.

Read	Overloaded. Read data from the server and store it in a byte array.
Reset	Reset the internal state of the object, resetting all properties to their default values.
SelectGroup	Selects the specified newsgroup as the current newsgroup.
StoreArticle	Overloaded. Retrieve an article from the selected newsgroup and store it in a file.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
Write	Overloaded. Write one or more bytes of data to the server.

Public Instance Events

OnCancel	Occurs when a blocking client operation is canceled.
OnCommand	Occurs when the client sends a command to the remote host and receives a reply indicating the result of that command.
OnConnect	Occurs when a connection is established with the remote host.
OnDisconnect	Occurs when the remote host disconnects from the local system.
OnError	Occurs when an client operation fails.
OnProgress	Occurs as a data stream is being read or written to the client.
OnRead	Occurs when data is available to be read from the client.
OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
OnWrite	Occurs when data can be written to the client.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the NntpClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

NntpClient Class | SocketTools Namespace

NntpClient Constructor

Initializes a new instance of the NntpClient class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public NntpClient();
```

See Also

NntpClient Class | SocketTools Namespace

NntpClient.NewsArticle Structure

This structure is used by the GetFirstArticle and GetNextArticle methods return information about a news article on the server.

For a list of all members of this type, see NntpClient.NewsArticle Members.

System.Object
  System.ValueType
    SocketTools.NntpClient.NewsArticle

[Visual Basic]
```
Public Structure NntpClient.NewsArticle
```

[C#]
```
public struct NntpClient.NewsArticle
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NntpClient (in SocketTools.NntpClient.dll)

See Also

NntpClient.NewsArticle Members | SocketTools Namespace

NntpClient.NewsArticle Members

NntpClient.NewsArticle overview

Public Instance Fields

Article	Get the news article number.
Author	Gets the author of the message.
DatePosted	Gets the date and time the news article was posted.
Lines	Gets the number of lines of text in the article.
MessageId	Gets the unique message ID for the news article.
References	Gets the references to the news article.
Size	Gets the size of the article in bytes.
Subject	Gets the subject of the news article.

Public Instance Methods

Equals (inherited from ValueType)	Indicates whether this instance and a specified object are equal.
GetHashCode (inherited from ValueType)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from ValueType)	Returns the fully qualified type name of this instance.

See Also

NntpClient.NewsArticle Class | SocketTools Namespace

NntpClient.NewsArticle Fields

The fields of the **NntpClient.NewsArticle** structure are listed below. For a complete list of **NntpClient.NewsArticle** structure members, see the NntpClient.NewsArticle Members topic.

Public Instance Fields

◆ Article	Get the news article number.
◆ Author	Gets the author of the message.
◆ DatePosted	Gets the date and time the news article was posted.
◆ Lines	Gets the number of lines of text in the article.
◆ MessageId	Gets the unique message ID for the news article.
◆ References	Gets the references to the news article.
◆ Size	Gets the size of the article in bytes.
◆ Subject	Gets the subject of the news article.

See Also

NntpClient.NewsArticle Class | SocketTools Namespace

NntpClient.NewsArticle.Article Field

Get the news article number.

[Visual Basic]
```
Public Article As Integer
```

[C#]
```
public int Article;
```

See Also

NntpClient.NewsArticle Class | SocketTools Namespace

NntpClient.NewsArticle.Author Field

Gets the author of the message.

[Visual Basic]

```
Public Author As String
```

[C#]

```
public string Author;
```

See Also

NntpClient.NewsArticle Class | SocketTools Namespace

NntpClient.NewsArticle.DatePosted Field

Gets the date and time the news article was posted.

[Visual Basic]

```
Public DatePosted As Date
```

[C#]

```
public DateTime DatePosted;
```

See Also

NntpClient.NewsArticle Class | SocketTools Namespace

NntpClient.NewsArticle.Lines Field

Gets the number of lines of text in the article.

[Visual Basic]

```
Public Lines As Integer
```

[C#]

```
public int Lines;
```

See Also

NntpClient.NewsArticle Class | SocketTools Namespace

NntpClient.NewsArticle.MessageId Field

Gets the unique message ID for the news article.

[Visual Basic]

```
Public MessageId As String
```

[C#]

```
public string MessageId;
```

See Also

NntpClient.NewsArticle Class | SocketTools Namespace

NntpClient.NewsArticle.References Field

Gets the references to the news article.

[Visual Basic]
```
Public References As String
```

[C#]
```
public string References;
```

See Also

NntpClient.NewsArticle Class | SocketTools Namespace

NntpClient.NewsArticle.Size Field

Gets the size of the article in bytes.

[Visual Basic]
```
Public Size As Integer
```

[C#]
```
public int Size;
```

See Also

NntpClient.NewsArticle Class | SocketTools Namespace

NntpClient.NewsArticle.Subject Field

Gets the subject of the news article.

[Visual Basic]

```
Public Subject As String
```

[C#]

```
public string Subject;
```

See Also

NntpClient.NewsArticle Class | SocketTools Namespace

NntpClient.NewsGroup Structure

This structure is used by the GetFirstGroup and GetNextGroup methods return information about a news article on the server.

For a list of all members of this type, see NntpClient.NewsGroup Members.

System.Object
  System.ValueType
    SocketTools.NntpClient.NewsGroup

[Visual Basic]
```
Public Structure NntpClient.NewsGroup
```

[C#]
```
public struct NntpClient.NewsGroup
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NntpClient (in SocketTools.NntpClient.dll)

See Also

NntpClient.NewsGroup Members | SocketTools Namespace

NntpClient.NewsGroup Members

NntpClient.NewsGroup overview

Public Instance Fields

Access	Gets the access permissions for the newsgroup.
FirstArticle	Gets the first available article number.
LastArticle	Gets the last available article number.
Name	Gets the name of the newsgroup.

Public Instance Methods

Equals (inherited from ValueType)	Indicates whether this instance and a specified object are equal.
GetHashCode (inherited from ValueType)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from ValueType)	Returns the fully qualified type name of this instance.

See Also

NntpClient.NewsGroup Class | SocketTools Namespace

NntpClient.NewsGroup Fields

The fields of the **NntpClient.NewsGroup** structure are listed below. For a complete list of **NntpClient.NewsGroup** structure members, see the NntpClient.NewsGroup Members topic.

Public Instance Fields

Access	Gets the access permissions for the newsgroup.
FirstArticle	Gets the first available article number.
LastArticle	Gets the last available article number.
Name	Gets the name of the newsgroup.

See Also

NntpClient.NewsGroup Class | SocketTools Namespace

NntpClient.NewsGroup.Access Field

Gets the access permissions for the newsgroup.

[Visual Basic]

```
Public Access As NewsGroupAccess
```

[C#]

```
public NewsGroupAccess Access;
```

See Also

NntpClient.NewsGroup Class | SocketTools Namespace

NntpClient.NewsGroup.FirstArticle Field

Gets the first available article number.

[Visual Basic]

```
Public FirstArticle As Integer
```

[C#]

```
public int FirstArticle;
```

See Also

NntpClient.NewsGroup Class | SocketTools Namespace

NntpClient.NewsGroup.LastArticle Field

Gets the last available article number.

[Visual Basic]

```
Public LastArticle As Integer
```

[C#]

```
public int LastArticle;
```

See Also

NntpClient.NewsGroup Class | SocketTools Namespace

NntpClient.NewsGroup.Name Field

Gets the name of the newsgroup.

[Visual Basic]

```
Public Name As String
```

[C#]

```
public string Name;
```

See Also

NntpClient.NewsGroup Class | SocketTools Namespace

NntpClient Properties

The properties of the **NntpClient** class are listed below. For a complete list of **NntpClient** class members, see the NntpClient Members topic.

Public Instance Properties

Article	Gets and sets the current article number in the selected newsgroup.
ArticleCount	Gets the number of available articles in the current newsgroup.
AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
Blocking	Gets and sets a value which indicates if the client is in blocking mode.
CertificateExpires	Get a value that specifies the date that the security certificate expires.
CertificateIssued	Get a value that specifies the date that the security certificate was issued.
CertificateIssuer	Get a value that provides information about the organization that issued the certificate.
CertificateName	Gets and sets a value that specifies the name of the client certificate.
CertificatePassword	Gets and sets the password associated with the client certificate.
CertificateStatus	Gets a value which indicates the status of the security certificate returned by the remote host.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateSubject	Gets a value that provides information about the organization that the server certificate was issued to.
CertificateUser	Gets and sets the user that owns the client certificate.
CipherStrength	Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.
FirstArticle	Gets the first available article in the selected newsgroup.
GroupName	Gets and sets the name of the currently selected newsgroup.
GroupTitle	Gets the title associated with the currently selected newsgroup.
Handle	Gets a value that specifies the client handle

	allocated for the current session.
HashStrength	Gets a value which specifies the length of the message digest that was selected for a secure connection.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsConnected	Gets a value which indicates if a connection to the remote host has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsReadable	Gets a value which indicates if there is data available to be read from the socket connection to the server.
IsWritable	Gets a value which indicates if data can be written to the client without blocking.
LastArticle	Gets the last available article in the selected newsgroup.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets the local Internet address that the client is bound to.
Localize	Gets a value that specifies if the date and time are localized.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
MessageId	Gets the unique message identifier for the current news article.
Options	Gets and sets a value which specifies one or more client options.
Password	Gets and sets the password used to authenticate the client session.
RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.

ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.
Secure	Gets and sets a value which specifies if a secure connection is established.
SecureCipher	Gets a value that specifies the encryption algorithm used for a secure connection.
SecureHash	Gets a value that specifies the message digest algorithm used for a secure connection.
SecureKeyExchange	Gets a value that specifies the key exchange algorithm used for a secure connection.
SecureProtocol	Gets and sets a value which specifies the protocol used for a secure connection.
Status	Gets a value which specifies the current status of the client.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
TimeZone	Gets and sets the current timezone offset in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
UserName	Gets and sets the username used to authenticate the client session.
Version	Gets a value which returns the current version of the NntpClient class library.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.Article Property

Gets and sets the current article number in the selected newsgroup.

[Visual Basic]

```
Public Property Article As Integer
```

[C#]

```
public int Article {get; set;}
```

Property Value

An integer value which specifies the article number.

Remarks

When a newsgroup is selected using the **SelectGroup** method, the value of this property is initialized to the first available news article in the group.

Setting the **Article** property updates the **MessageId** property to reflect the specified article's message ID.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.ArticleCount Property

Gets the number of available articles in the current newsgroup.

[Visual Basic]
```
Public ReadOnly Property ArticleCount As Integer
```

[C#]
```
public int ArticleCount {get;}
```

Property Value

An integer value which specifies the number of available news articles.

Remarks

This property value is only meaningful after the **SelectGroup** method has been called.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.AutoResolve Property

Gets and sets a value that determines if host names and addresses are automatically resolved.

[Visual Basic]

```
Public Property AutoResolve As Boolean
```

[C#]

```
public bool AutoResolve {get; set;}
```

Property Value

Returns **true** if host names are automatically resolved to Internet addresses. The default value is **false**.

Remarks

Setting the **AutoResolve** property determines if the class automatically resolves host names and addresses specified by the **HostName** and **HostAddress** properties. If set to **true**, setting the **HostName** property will cause the class to automatically determine the corresponding IP address and update the **HostAddress** property accordingly. Likewise, setting the **HostAddress** property will cause the class to determine the host name and update the **HostName** property. Setting this property to **false** prevents the class from resolving host names until a connection attempt is made.

It is important to note that setting the **HostName** or **HostAddress** property may cause the current thread to block, sometimes for several seconds, until the name or address is resolved. To prevent this behavior, set this property value to **false**.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.Blocking Property

Gets and sets a value which indicates if the client is in blocking mode.

[Visual Basic]

```
Public Property Blocking As Boolean
```

[C#]

```
public bool Blocking {get; set;}
```

Property Value

Returns **true** if the client is in blocking mode; otherwise it returns **false**. The default value is **true**.

Remarks

Setting the **Blocking** property determines if client operations complete synchronously or asynchronously. If set to **true**, then each client operation (such as sending or receiving data) will return when the operation has completed or timed-out. If set to **false**, client operations will return immediately. If the operation would result in the client blocking (such as attempting to read data when no data has been sent by the remote host), an error is generated.

It is important to note that certain events, such as **OnDisconnect, OnRead** and **OnWrite** are only fired if the client is in non-blocking mode.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.CertificateExpires Property

Get a value that specifies the date that the security certificate expires.

[Visual Basic]

```
Public ReadOnly Property CertificateExpires As String
```

[C#]

```
public string CertificateExpires {get;}
```

Property Value

A string which specifies a date using the local date and time format.

Remarks

The **CertificateExpires** property returns a string that specifies the date and time that the security certificate expires. This property will return an empty string if a secure connection has not been established with the remote host.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.CertificateIssued Property

Get a value that specifies the date that the security certificate was issued.

[Visual Basic]

```
Public ReadOnly Property CertificateIssued As String
```

[C#]

```
public string CertificateIssued {get;}
```

Property Value

A string which specifies a date using the local date and time format.

Remarks

The **CertificateIssued** property returns a string that specifies the date and time that the security certificate was issued. This property will return an empty string if a secure connection has not been established with the remote host.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.CertificateIssuer Property

Get a value that provides information about the organization that issued the certificate.

[Visual Basic]

```
Public ReadOnly Property CertificateIssuer As String
```

[C#]

```
public string CertificateIssuer {get;}
```

Property Value

A string that contains a comma separated list of name value pairs.

Remarks

The **CertificateIssuer** property returns a string that contains information about the organization that issued the server certificate. The string value is a comma separated list of tagged name and value pairs. In the nomenclature of the X.500 standard, each of these pairs are called a relative distinguished name (RDN), and when concatenated together, forms the issuer's distinguished name (DN). For example:

```
C=US, O="RSA Data Security, Inc.", OU=Secure Server Certification Authority
```

To obtain a specific value, such as the name of the issuer or the issuer's country, the application must parse the string returned by this property. Some of the common tokens used in the distinguished name are:

Token	Description
C	The ISO standard two character country code.
S	The name of the state or province.
L	The name of the city or locality.
O	The name of the company or organization.
OU	The name of the department or organizational unit
CN	The common name; with X.509 certificates, this is the domain name of the site the certificate was issued for.

This property will return an empty string if a secure connection has not been established with the remote host.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.CertificateName Property

Gets and sets a value that specifies the name of the client certificate.

[Visual Basic]

```
Public Property CertificateName As String
```

[C#]

```
public string CertificateName {get; set;}
```

Property Value

A string which specifies the certificate name.

Remarks

The **CertificateName** property is used to specify the name of a client certificate to use when establishing a secure connection. It is only required that you set this property value if the server requires a client certificate for authentication. If this property is not set, a client certificate will not be provided to the server. If a certificate name is specified, the certificate must have a private key associated with it, otherwise the connection attempt will fail because the control will be unable to create a security context for the session.

When the certificate store is searched for a matching certificate, it will first search for any certificate with a friendly name that matches the property value. If no valid certificate is found, it will then search for a certificate with a matching common name.

Certificates may be installed and viewed on the local system using the Certificate Manager that is included with the Windows operating system. For more information, refer to the documentation for the Microsoft Management Console.

See Also

NntpClient Class | SocketTools Namespace | CertificateStore Property | Secure Property

NntpClient.CertificateStatus Property

Gets a value which indicates the status of the security certificate returned by the remote host.

[Visual Basic]

```
Public ReadOnly Property CertificateStatus As SecurityCertificate
```

[C#]

```
public NntpClient.SecurityCertificate CertificateStatus {get;}
```

Property Value

A SecurityCertificate enumeration value which specifies the status of the certificate.

Remarks

The **CertificateStatus** property is used to determine the status of the security certificate returned by the remote host when a secure connection has been established. This property value should be checked after the connection to the server has completed, but prior to beginning a transaction.

Note that if the certificate cannot be validated, the secure connection will not be automatically terminated. It is the responsibility of your application to determine the best course of action to take if the certificate is invalid. Even if the security certificate cannot be validated, the data exchanged with the remote host will still be encrypted.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.CertificateStore Property

Gets and sets a value that specifies the name of the local certificate store.

[Visual Basic]
```
Public Property CertificateStore As String
```

[C#]
```
public string CertificateStore {get; set;}
```

Property Value

A string which specifies the certificate store name. The default value is the current user's personal certificate store.

Remarks

The **CertificateStore** property is used to specify the name of the certificate store which contains the security certificate to use when establishing a secure connection. The certificate may either be stored in the registry or in a file. If the certificate is stored in the registry, then this property should be set to one of the following predefined values:

Store Name	Description
CA	Certification authority certificates. These are certificates that are issued by entities which are entrusted to issue certificates to other individuals or organizations. Companies such as VeriSign and Thawte act as certification authorities.
MY	Personal certificates and their associated private keys for the current user. This store typically holds the client certificates used to establish a user's credentials. If a certificate store is not specified, this is the default value that is used.
ROOT	Certificates that have been self-signed by a certificate authority. Root certificates for a number of different certification authorities such as VeriSign and Thawte are installed as part of the operating system and periodically updated by Microsoft.

In most cases the client certificate will be installed in the user's personal certificate store, and therefore it is not necessary to set this property value because that is the default location that will be used to search for the certificate. This property is only used if the **CertificateName** property is also set to a valid certificate name.

If you are using a local certificate store, with the certificate and private key stored in the registry, you can explicitly specify whether the certificate store for the current user or the local machine (all users) should be used. This is done by prefixing the certificate store name with "HKCU" for the current user, or "HKLM" for the local machine. For example, a certificate store name of "HKLM:MY" would specify the personal certificate store for the local machine, rather than the current user. If neither prefix is specified, it will default to the certificate store for the current user.

This property may also be used to specify a file that contains the client certificate. In this case, the property should specify the full path to the file and must contain both the certificate and private key in PKCS12

format. If the file is protected by a password, the **CertificatePassword** property must also be set to specify the password.

See Also

NntpClient Class | SocketTools Namespace | CertificatePassword Property | Secure Property

NntpClient.CertificateSubject Property

Gets a value that provides information about the organization that the server certificate was issued to.

[Visual Basic]

```
Public ReadOnly Property CertificateSubject As String
```

[C#]

```
public string CertificateSubject {get;}
```

Property Value

A string that contains a comma separated list of name value pairs.

Remarks

The **CertificateSubject** property returns a string that contains information about the organization that the server certificate was issued to. The string value is a comma separated list of tagged name and value pairs. In the nomenclature of the X.500 standard, each of these pairs are called a relative distinguished name (RDN), and when concatenated together, forms the issuer's distinguished name (DN). For example:

```
C=US, O="RSA Data Security, Inc.", OU=Secure Server Certification Authority
```

To obtain a specific value, such as the name of the issuer or the issuer's country, the application must parse the string returned by this property. Some of the common tokens used in the distinguished name are:

Token	Description
C	The ISO standard two character country code.
S	The name of the state or province.
L	The name of the city or locality.
O	The name of the company or organization.
OU	The name of the department or organizational unit
CN	The common name; with X.509 certificates, this is the domain name of the site the certificate was issued for.

This property will return an empty string if a secure connection has not been established with the remote host.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.CipherStrength Property

Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.

[Visual Basic]

```
Public ReadOnly Property CipherStrength As Integer
```

[C#]

```
public int CipherStrength {get;}
```

Property Value

An integer value which specifies the encryption key length if a secure connection has been established; otherwise a value of 0 is returned.

Remarks

The **CipherStrength** property returns the number of bits in the key used to encrypt the secure data stream. Common values returned by this property are 128 and 256. A key length of 40 or 56 bits is considered insecure and subject to brute force attacks. 128-bit and 256-bit keys are considered secure. If this property returns a value of 0, this means that a secure connection has not been established with the remote host.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.FirstArticle Property

Gets the first available article in the selected newsgroup.

[Visual Basic]
```
Public Property FirstArticle As Integer
```

[C#]
```
public int FirstArticle {get; set;}
```

Property Value

An integer value which specifies the first article number.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.GroupName Property

Gets and sets the name of the currently selected newsgroup.

[Visual Basic]

```
Public Property GroupName As String
```

[C#]

```
public string GroupName {get; set;}
```

Property Value

A string which specifies the current newsgroup name.

Remarks

Setting this property value is similar to calling the **SelectGroup** method. The current newsgroup will change to the specified newsgroup and the current article will be changed to the first available article in the group.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.GroupTitle Property

Gets the title associated with the currently selected newsgroup.

[Visual Basic]

```
Public ReadOnly Property GroupTitle As String
```

[C#]

```
public string GroupTitle {get;}
```

Property Value

A string describing the currently selected newsgroup.

Remarks

The news server must support the XGTITLE command so that the group description can be obtained when the newsgroup is selected. If this command is not recognized, then no description will be returned.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.Handle Property

Gets a value that specifies the client handle allocated for the current session.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer which represents a client handle. If there is no active connection, a value of -1 is returned.

Remarks

The **Handle** property specifies the numeric descriptor of the current client session.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.HashStrength Property

Gets a value which specifies the length of the message digest that was selected for a secure connection.

[Visual Basic]

```
Public ReadOnly Property HashStrength As Integer
```

[C#]

```
public int HashStrength {get;}
```

Property Value

An integer value which specifies the length of the message digest if a secure connection has been established; otherwise a value of 0 is returned.

Remarks

The **HashStrength** property returns the number of bits used in the message digest (hash) that was selected. Common values returned by this property are 128 and 160. If this property returns a value of 0, this means that a secure connection has not been established with the remote host.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.HostAddress Property

Gets and sets a value which specifies the Internet address used to establish a connection.

[Visual Basic]

```
Public Property HostAddress As String
```

[C#]

```
public string HostAddress {get; set;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **HostAddress** property can be used to set the Internet address for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the address is assigned to a valid host name, the **HostName** property will be updated with that value.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.HostName Property

Gets and sets a value which specifies the host name used to establish a connection.

[Visual Basic]

```
Public Property HostName As String
```

[C#]

```
public string HostName {get; set;}
```

Property Value

A string which specifies a host name.

Remarks

The **HostName** property can be used to set the host name for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the name can be resolved to a valid Internet address, the **HostAddress** property will be updated with that value.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.IsBlocked Property

Gets a value which indicates if the current thread is performing a blocking client operation.

[Visual Basic]
```
Public ReadOnly Property IsBlocked As Boolean
```

[C#]
```
public bool IsBlocked {get;}
```

Property Value

Returns **true** if the current thread is blocking, otherwise returns **false**.

Remarks

The **IsBlocked** property returns **true** if the current thread is blocked performing an operation. Because the Windows Sockets API only permits one blocking operation per thread of execution, this property should be checked before starting any blocking operation in response to an event.

If the **IsBlocked** property returns **false**, this means there are no blocking operations on the current thread at that time. However, this does not guarantee that the next client operation will not fail. An application should always check the return value from a client operation and check the value of the **LastError** property if an error occurs.

Note that this property will return **true** if there is any blocking operation being performed by the current thread, regardless of whether this specific instance of the class is responsible for the blocking operation or not.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.IsConnected Property

Gets a value which indicates if a connection to the remote host has been established.

[Visual Basic]
```
Public ReadOnly Property IsConnected As Boolean
```

[C#]
```
public bool IsConnected {get;}
```

Property Value

Returns **true** if the connection has been established; otherwise returns **false**.

Remarks

The **IsConnected** property can only be used to indicate if there is still a logical connection to the remote host. It cannot be used to detect abnormal conditions such as the remote host aborting the connection, the physical network connection being lost or other critical errors.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.IsInitialized Property

Gets a value which indicates if the current instance of the class has been initialized successfully.

[Visual Basic]

```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]

```
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the class instance has been initialized; otherwise returns **false**.

Remarks

The **IsInitialized** property is used to determine if the current instance of the class has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the class may not be able to initialize itself.

The most common reasons that a class instance may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required networking libraries or not being able to access the system registry.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.IsReadable Property

Gets a value which indicates if there is data available to be read from the socket connection to the server.

[Visual Basic]

```
Public ReadOnly Property IsReadable As Boolean
```

[C#]

```
public bool IsReadable {get;}
```

Property Value

Returns **true** if there is data available to be read; otherwise returns **false**.

Remarks

The **IsReadable** property returns **true** if data can be read from the client without blocking. For non-blocking sessions, this property can be checked before the application attempts to read the client. Note that even if this property does return **true** indicating that there is data available to be read, applications should always check the return value from the **Read** method.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.IsWritable Property

Gets a value which indicates if data can be written to the client without blocking.

[Visual Basic]
```
Public ReadOnly Property IsWritable As Boolean
```

[C#]
```
public bool IsWritable {get;}
```

Property Value

Returns **true** if data can be written to the client; otherwise returns **false**.

Remarks

The **IsWritable** property returns **true** if data can be written to the client without blocking. For non-blocking sessions, this property can be checked before the application attempts to write data to the client. Note that even if this property does return **true** indicating that data can be written to the client, applications should always check the return value from the **Write** method.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.LastArticle Property

Gets the last available article in the selected newsgroup.

[Visual Basic]

```
Public Property LastArticle As Integer
```

[C#]

```
public int LastArticle {get; set;}
```

Property Value

An integer value which specifies the last article number.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public NntpClient.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.LocalAddress Property

Gets the local Internet address that the client is bound to.

[Visual Basic]
```
Public ReadOnly Property LocalAddress As String
```

[C#]
```
public string LocalAddress {get;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **LocalAddress** property returns the local Internet address that the client is bound to when a connection is established with a remote host. This property may return either an IPv4 or IPv6 formatted address, depending on the type of connection that was established.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.Localize Property

Gets a value that specifies if the date and time are localized.

[Visual Basic]

```
Public Property Localize As Boolean
```

[C#]

```
public bool Localize {get; set;}
```

Property Value

A boolean value which specifies if the date and time is localized.

Remarks

Setting the **Localize** property controls how date and time values are localized. If the property is set to **true**, then the date and time will be adjusted to the current timezone. If the property is set to **false**, the date and time are returned as UTC (Coordinated Universal Time) values.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.LocalName Property

Gets a value which specifies the host name for the local system.

[Visual Basic]

```
Public ReadOnly Property LocalName As String
```

[C#]

```
public string LocalName {get;}
```

Property Value

A string which specifies the local host name.

Remarks

The **LocalName** property returns the fully-qualified host name assigned to the local system. If the system has not been configured with an Internet domain name, then this property will return the NetBIOS name assigned to the local system.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.LocalPort Property

Gets the local port number the client is bound to.

[Visual Basic]
```
Public ReadOnly Property LocalPort As Integer
```

[C#]
```
public int LocalPort {get;}
```

Property Value

An integer value which specifies a port number. The default value is 0.

Remarks

The **LocalPort** property is used to identify the local port number that the client is bound to to when a connection is established with a remote host.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.MessageId Property

Gets the unique message identifier for the current news article.

[Visual Basic]

```
Public Property MessageId As String
```

[C#]

```
public string MessageId {get; set;}
```

Property Value

A string that specifies the unique message identifier for the current article.

Remarks

Setting this property will cause the current article number to change to match the article that the message identifier specifies. If no such article exists, an exception will be generated.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.Options Property

Gets and sets a value which specifies one or more client options.

[Visual Basic]

```
Public Property Options As NewsOptions
```

[C#]

```
public NntpClient.NewsOptions Options {get; set;}
```

Property Value

Returns one or more NewsOptions enumeration flags which specify the options for the client. The default value for this property is **NewsOptions.optionNone**.

Remarks

The **Options** property specifies one or more default options options which are used when establishing a connection using the **Connect** method.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.Password Property

Gets and sets the password used to authenticate the client session.

[Visual Basic]

```
Public Property Password As String
```

[C#]

```
public string Password {get; set;}
```

Property Value

A string which specifies the password.

Remarks

If a password is not specified when the **Connect** method is called, the value of this property will be used as the default password when establishing a connection with the server.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.RemotePort Property

Gets and sets a value which specifies the remote port number.

[Visual Basic]

```
Public Property RemotePort As Integer
```

[C#]

```
public int RemotePort {get; set;}
```

Property Value

An integer value which specifies a port number.

Remarks

The **RemotePort** property is used to set the port number that will be used to establish a connection with a remote host. If the port number specifies a well-known port, the **RemoteService** property will be updated with that name.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.ResultCode Property

Gets a value which specifies the last result code returned by the server.

[Visual Basic]

```
Public ReadOnly Property ResultCode As Integer
```

[C#]

```
public int ResultCode {get;}
```

Property Value

An integer value which specifies the last result code returned by the server.

Remarks

Result codes are three-digit numeric values returned by the remote server and may be broken down into the following ranges:

ResultCode	Description
100-199	Positive preliminary result. This indicates that the requested action is being initiated, and the client should expect another reply from the server before proceeding.
200-299	Positive completion result. This indicates that the server has successfully completed the requested action.
300-399	Positive intermediate result. This indicates that the requested action cannot complete until additional information is provided to the server.
400-499	Transient negative completion result. This indicates that the requested action did not take place, but the error condition is temporary and may be attempted again.
500-599	Permanent negative completion result. This indicates that the requested action did not take place.

It is important to note that while some result codes have become standardized, not all servers respond to commands using the same result codes. For example, one server may respond with a result code of 221 to indicate success, while another may respond with a value of 235. It is recommended that applications check for ranges of values to determine if a command was successful, not a specific value.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.ResultString Property

Gets a string value which describes the result of the previous command.

[Visual Basic]

```
Public ReadOnly Property ResultString As String
```

[C#]

```
public string ResultString {get;}
```

Property Value

A string which describes the result of the previous command executed on the server.

Remarks

The **ResultString** property returns the result string from the last action taken by the client. This string is generated by the remote server, and typically is used to describe the result code. For example, if an error is indicated by the result code, the result string may describe the condition that caused the error.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.Secure Property

Gets and sets a value which specifies if a secure connection is established.

[Visual Basic]

```
Public Property Secure As Boolean
```

[C#]

```
public bool Secure {get; set;}
```

Property Value

Returns **true** if a secure connection is established; otherwise returns **false**. The default value is **false**.

Remarks

The **Secure** property determines if a secure connection is established with the remote host. The default value for this property is **false**, which specifies that a standard connection to the server is used. To establish a secure connection, the application should set this property value to **true** prior to calling the **Connect** method. Once the connection has been established, the client may exchange data with the server as with standard connections.

It is strongly recommended that any application that sets this property **true** use error handling to trap an errors that may occur. If the control is unable to initialize the security libraries, or otherwise cannot create a secure session for the client, an exception may be generated when this property value is set.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.SecureCipher Property

Gets a value that specifies the encryption algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureCipher As SecureCipherAlgorithm
```

[C#]

```
public NntpClient.SecureCipherAlgorithm SecureCipher {get;}
```

Property Value

A SecureCipherAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureCipher** property returns a value which identifies the algorithm used to encrypt the data stream. If a secure connection has not been established, this property will return a value of **cipherNone**.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.SecureHash Property

Gets a value that specifies the message digest algorithm used for a secure connection.

[Visual Basic]
```
Public ReadOnly Property SecureHash As SecureHashAlgorithm
```

[C#]
```
public NntpClient.SecureHashAlgorithm SecureHash {get;}
```

Property Value

A SecureHashAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureHash** property returns a value which identifies the message digest (hash) algorithm that was selected when a secure connection was established. If a secure connection has not been established, this property will return a value of **hashNone**.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.SecureKeyExchange Property

Gets a value that specifies the key exchange algorithm used for a secure connection.

[Visual Basic]
```
Public ReadOnly Property SecureKeyExchange As SecureKeyAlgorithm
```

[C#]
```
public NntpClient.SecureKeyAlgorithm SecureKeyExchange {get;}
```

Property Value

A SecureKeyAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureKeyExchange** property returns a value which identifies the key exchange algorithm that was selected when a secure connection was established. If a secure connection has not been established, this property will return a value of **keyExchangeNone**.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.SecureProtocol Property

Gets and sets a value which specifies the protocol used for a secure connection.

[Visual Basic]
```
Public Property SecureProtocol As SecurityProtocols
```

[C#]
```
public NntpClient.SecurityProtocols SecureProtocol {get; set;}
```

Property Value

A SecurityProtocols enumeration value which identifies the protocol to be used when establishing a secure connection.

Remarks

The **SecureProtocol** property can be used to specify the security protocol to be used when establishing a secure connection with a server or accepting a secure connection from a client. By default, the class will attempt to use either SSL v3 or TLS v1 to establish the connection, with the appropriate protocol automatically selected based on the capabilities of the remote host. It is recommended that you only change this property value if you fully understand the implications of doing so. Assigning a value to this property will override the default protocol and force the class to attempt to use only the protocol specified.

Multiple security protocols may be specified by combining them using a bitwise **or** operator. After a connection has been established, this property will identify the protocol that was selected. Attempting to set this property after a connection has been established will result in an exception being thrown. This property should only be set after setting the **Secure** property to **true** and before calling the **Accept** or **Connect** methods.

In some cases, a server may only accept a secure connection if the TLS v1 protocol is specified. If the security protocol is not compatible with the server, then the connection will fail with an error indicating that the control is unable to establish a security context for the session. In this case, try assigning the property to **protocolTLS1** and attempt the connection again.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.Status Property

Gets a value which specifies the current status of the client.

[Visual Basic]

```
Public ReadOnly Property Status As NewsStatus
```

[C#]

```
public NntpClient.NewsStatus Status {get;}
```

Property Value

A NewsStatus enumeration value which specifies the current client status.

Remarks

The **Status** property returns the current status of the client. This property can be used to check on blocking connections to determine if the client is interacting with the remote host before taking some action.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.ThreadModel Property

Gets and sets a value which specifies the threading model for the class instance.

[Visual Basic]
```
Public Property ThreadModel As ThreadingModel
```

[C#]
```
public NntpClient.ThreadingModel ThreadModel {get; set;}
```

Property Value

Returns one or more ThreadingModel enumeration value which specifies the threading model for the client. The default value for this property is **modelSingleThread**.

Remarks

The **ThreadModel** property specifies the threading model that is used by the class instance when a connection is established. The default value for this property is **modelSingleThread**, which specifies that only the thread that established the connection should be permitted to invoke methods. It is important to note that this threading model does not limit the application to a single thread of execution. When a session is established using the **Connect** method, that session is attached to the thread that created it. From that point on, until the session is terminated, only the owner may invoke methods in that instance of the class. The ownership of the class instance may be transferred from one thread to another using the **AttachThread** method.

Setting this property to **modelFreeThread** disables certain internal safety checks that are performed by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads. The application must ensure that no two threads will attempt to invoke a blocking method at the same time. In other words, if one thread invokes a method, the application must ensure that another thread will not attempt to invoke any other method at the same time using the same instance of the class.

Changing the value of this property will not affect an active client session. The threading model must be specified prior to invoking the **Connect** method.

See Also

NntpClient Class | SocketTools Namespace | AttachThread Method | ThreadingModel Enumeration | ThreadModel Attribute

NntpClient.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]

```
Public Property ThrowError As Boolean
```

[C#]

```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.Timeout Property

Gets and sets a value which specifies a timeout period in seconds.

[Visual Basic]

```
Public Property Timeout As Integer
```

[C#]

```
public int Timeout {get; set;}
```

Property Value

An integer value which specifies a timeout period in seconds.

Remarks

Setting the **Timeout** property specifies the number of seconds until a blocking operation fails and returns an error.

The timeout period is only used when the client is in blocking mode. Although this property can be changed when the client is in non-blocking mode, the value will be ignored until the client is returned to blocking mode.

For most applications it is recommended the timeout period be set between 10 and 20 seconds.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.TimeZone Property

Gets and sets the current timezone offset in seconds.

[Visual Basic]

```
Public Property TimeZone As Integer
```

[C#]

```
public int TimeZone {get; set;}
```

Property Value

An integer value which specifies the current timezone offset in seconds.

Remarks

The **TimeZone** property returns the current offset from UTC in seconds. Setting the property changes the current timezone offset to the specified value. The value of this property is initially determined by the date and time settings on the local system.

This property value is used in conjunction with the **Localize** property to control how date and time localization is handled.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.Trace Property

Gets and sets a value which indicates if network function logging is enabled.

[Visual Basic]
```
Public Property Trace As Boolean
```

[C#]
```
public bool Trace {get; set;}
```

Property Value

Returns **true** if network function tracing is enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Trace** property is used to enable (or disable) the tracing of network function calls. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the SocketTools networking classes and is enabled or disabled for an entire process. This means that once trace logging is enabled for a given component, all of the function calls made by the process using any of the SocketTools classes will be logged. For example, if you have an application using both the File Transfer Protocol and Post Office Protocol classes, and you set the **Trace** property to **true**, function calls made by both classes will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all classes used by the process.

If trace logging is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the logfile is fairly large. Since logfiles can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

When redistributing your application, make sure that you include the **SocketTools10.TraceLog.dll** module with your installation. If this library is not present, then no trace output will be generated and the value of the **Trace** property will be ignored. Only those function calls made by the SocketTools networking classes will be logged. Calls made directly to the Windows Sockets API, or calls made by other classes, will not be logged.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.TraceFile Property

Gets and sets a value which specifies the name of the logfile.

[Visual Basic]
```
Public Property TraceFile As String
```

[C#]
```
public string TraceFile {get; set;}
```

Property Value

A string which specifies the name of the file.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named **SocketTools.log** is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFile** property should be set to the same value for each instance. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
 MyApp INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0 MyApp WRN:
connect(46, 192.0.0.1:1234, 16) returned -1 [10035] MyApp ERR: accept(46,
NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the control is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Note that if the specified file cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.TraceFlags Property

Gets and sets a value which specifies the client function tracing flags.

[Visual Basic]

```
Public Property TraceFlags As TraceOptions
```

[C#]

```
public NntpClient.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration which specifies the amount of detail written to the trace logfile.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Because network function tracing is enabled per-process, the trace flags are shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFlags** property should be set to the same value for each instance. Changing the trace flags for any one instance of the class will affect the logging performed for all SocketTools classes used by the application.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being written and the error **errorOperationWouldBlock** occurs, a warning is generated because the application simply needs to attempt to write the data at a later time.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.UserName Property

Gets and sets the username used to authenticate the client session.

[Visual Basic]

```
Public Property UserName As String
```

[C#]

```
public string UserName {get; set;}
```

Property Value

A string which specifies the username.

Remarks

If a username is not specified when the **Connect** method is called, the value of this property will be used as the default username when establishing a connection with the server.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.Version Property

Gets a value which returns the current version of the NntpClient class library.

[Visual Basic]

```
Public ReadOnly Property Version As String
```

[C#]

```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the NntpClient class library. This value can be used by an application for validation and debugging purposes.

See Also

NntpClient Class | SocketTools Namespace

NntpClient Methods

The methods of the **NntpClient** class are listed below. For a complete list of **NntpClient** class members, see the NntpClient Members topic.

Public Instance Methods

AttachThread	Attach an instance of the class to the current thread
Authenticate	Overloaded. Authenticate the client session with a username and password.
Cancel	Cancel the current blocking client operation.
CloseArticle	Closes the current article that has been opened or created.
Command	Overloaded. Send a custom command to the server.
Connect	Overloaded. Establish a connection with a remote host.
CreateArticle	Creates a new article in the current newsgroup.
Disconnect	Terminate the connection with a remote host.
Dispose	Overloaded. Releases all resources used by NntpClient.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetArticle	Overloaded. Retrieve an article from the server.
GetFirstArticle	Overloaded. Return information about a selected range of articles in the current newsgroup.
GetFirstGroup	Overloaded. Return information about newsgroups created after the specified date.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetHeaders	Overloaded. Retrieves the headers for the specified article from the server.
GetNextArticle	Return information about the next article available in the current newsgroup.
GetNextGroup	Return information about the next available newsgroup.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the NntpClient class.
OpenArticle	Overloaded. Opens the specified article in the currently selected newsgroup.

PostArticle	Overloaded. Post a new article to the currently selected newsgroup.
Read	Overloaded. Read data from the server and store it in a byte array.
Reset	Reset the internal state of the object, resetting all properties to their default values.
SelectGroup	Selects the specified newsgroup as the current newsgroup.
StoreArticle	Overloaded. Retrieve an article from the selected newsgroup and store it in a file.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
Write	Overloaded. Write one or more bytes of data to the server.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the NntpClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.AttachThread Method

Attach an instance of the class to the current thread

[Visual Basic]

```
Public Function AttachThread() As Boolean
```

[C#]

```
public bool AttachThread();
```

Return Value

A boolean value which specifies if the client could be attached to the current thread. If this method returns **false**, the client could not be attached to the thread and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

When an instance of the class is created it is associated with the current thread that created it. Normally, if another thread attempts to perform an operation using that instance, an error is returned. This is used to ensure that other threads cannot interfere with an operation being performed by the owner thread. In some cases, it may be desirable for one thread in an application to create an instance of the class, establish a connection and then pass that instance to another worker thread. The **AttachThread** method can be used to change the ownership of the class instance to the new worker thread.

This method should be called by the new thread immediately after it has been created, and if the new thread does not release the handle itself, the ownership of the handle should be restored by the original thread. Under no circumstances should **AttachThread** be used to forcibly destroy an instance of a class allocated by another thread while a blocking operation is in progress. To cancel a blocking operation, use the **Cancel** method and then delete the class instance after the blocking function exits and control is returned to the current thread.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.Authenticate Method

Authenticate the client session.

Overload List

Authenticate the client session.

public bool Authenticate();

Authenticate the client session with a username and password.

public bool Authenticate(string,string);

See Also

NntpClient Class | SocketTools Namespace

NntpClient.Authenticate Method ()

Authenticate the client session.

[Visual Basic]

```
Overloads Public Function Authenticate() As Boolean
```

[C#]

```
public bool Authenticate();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Authenticate** method identifies the client to the news server, which may be required to access certain newsgroups or to post articles. If the user name or password is invalid, an error will occur. This method should only be used if the server indicates that authentication is required by returning an error.

The value of the **UserName** property is used to specify the username and the value of the **Password** property is used to specify the password.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Authenticate Overload List

NntpClient.Authenticate Method (String, String)

Authenticate the client session with a username and password.

[Visual Basic]

```
Overloads Public Function Authenticate( _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]

```
public bool Authenticate(
   string userName,
   string userPassword
);
```

Parameters

userName
A string which specifies the username used to authenticate the client session.

userPassword
A string which specifies the password used to authenticate the client session.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Authenticate** method identifies the client to the news server, which may be required to access certain newsgroups or to post articles. If the user name or password is invalid, an error will occur. This method should only be used if the server indicates that authentication is required by returning an error.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Authenticate Overload List

NntpClient.Cancel Method

Cancel the current blocking client operation.

[Visual Basic]

```
Public Sub Cancel()
```

[C#]

```
public void Cancel();
```

Remarks

When the **Cancel** method is called, the blocking client operation will not immediately fail. An internal flag is set which causes the blocking operation to exit with an error. This means that the application cannot cancel an operation and immediately perform some other blocking function. Instead it must allow the calling stack to unwind, returning back to the blocking operation before making any further function calls.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.CloseArticle Method

Closes the current article that has been opened or created.

[Visual Basic]

```
Public Function CloseArticle() As Boolean
```

[C#]

```
public bool CloseArticle();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CloseArticle** method closes the current article that has been opened or created. If an article is being created, this method actually submits the article to the server. Note that the client application is responsible for generating the message headers as well as the body of the message. News articles conform to the same general characteristics of an email message.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.Command Method

Send a custom command to the server.

Overload List

Send a custom command to the server.

public bool Command(string);

Send a custom command to the server.

public bool Command(string,bool);

Send a custom command to the server.

public bool Command(string,string);

Send a custom command to the server.

public bool Command(string,string,bool);

See Also

NntpClient Class | SocketTools Namespace

NntpClient.Command Method (String)

Send a custom command to the server.

[Visual Basic]

```
Overloads Public Function Command( _
   ByVal command As String _
) As Boolean
```

[C#]

```
public bool Command(
   string command
);
```

Parameters

command

A string which specifies the command to send. Valid commands vary based on the Internet protocol and the type of server that the client is connected to. Consult the protocol standard and/or the technical reference documentation for the server to determine what commands may be issued by a client application.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Command** method sends a command to the remote host and processes the result code sent back in response to that command. This method can be used to send custom commands to a server to take advantage of features or capabilities that may not be supported internally by the class library.

To determine the specific status code returned by the server, check the value of the **ResultCode** property after the method returns.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Command Overload List

NntpClient.Command Method (String, Boolean)

Send a custom command to the server.

[Visual Basic]

```
Overloads Public Function Command( _
   ByVal command As String, _
   ByVal isMultiLine As Boolean _
) As Boolean
```

[C#]

```
public bool Command(
   string command,
   bool isMultiLine
);
```

Parameters

command

A string which specifies the command to send. Valid commands vary based on the Internet protocol and the type of server that the client is connected to. Consult the protocol standard and/or the technical reference documentation for the server to determine what commands may be issued by a client application.

isMultiLine

A boolean value which specifies if the command will result in multiple lines of output from the server. For more information about a specific command, consult the standards documentation for the protocol.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Command** method sends a command to the remote host and processes the result code sent back in response to that command. This method can be used to send custom commands to a server to take advantage of features or capabilities that may not be supported internally by the class library.

The *isMultiLine* parameter is used by the method to determine if multiple lines of data will be returned by the server as the result of a command. Unlike a single line response, which consists of a result code and result string, a multi-line response consists of one or more lines of text, terminated by a special end-of-data marker.

To determine the specific status code returned by the server, check the value of the **ResultCode** property after the method returns.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Command Overload List

NntpClient.Command Method (String, String)

Send a custom command to the server.

[Visual Basic]

```
Overloads Public Function Command( _
   ByVal command As String, _
   ByVal parameters As String _
) As Boolean
```

[C#]

```
public bool Command(
   string command,
   string parameters
);
```

Parameters

command

A string which specifies the command to send. Valid commands vary based on the Internet protocol and the type of server that the client is connected to. Consult the protocol standard and/or the technical reference documentation for the server to determine what commands may be issued by a client application.

parameters

An string which specifies one or more parameters to be sent along with the command. If more than one parameter is required, they must be separated by a single space character. Consult the protocol standard and/or technical reference documentation for the server to determine what parameters should be provided when issuing a specific command.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Command** method sends a command to the remote host and processes the result code sent back in response to that command. This method can be used to send custom commands to a server to take advantage of features or capabilities that may not be supported internally by the class library.

To determine the specific status code returned by the server, check the value of the **ResultCode** property after the method returns.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Command Overload List

NntpClient.Command Method (String, String, Boolean)

Send a custom command to the server.

[Visual Basic]

```
Overloads Public Function Command( _
   ByVal command As String, _
   ByVal parameters As String, _
   ByVal isMultiLine As Boolean _
) As Boolean
```

[C#]

```
public bool Command(
   string command,
   string parameters,
   bool isMultiLine
);
```

Parameters

command

A string which specifies the command to send. Valid commands vary based on the Internet protocol and the type of server that the client is connected to. Consult the protocol standard and/or the technical reference documentation for the server to determine what commands may be issued by a client application.

parameters

An string which specifies one or more parameters to be sent along with the command. If more than one parameter is required, they must be separated by a single space character. Consult the protocol standard and/or technical reference documentation for the server to determine what parameters should be provided when issuing a specific command.

isMultiLine

A boolean value which specifies if the command will result in multiple lines of output from the server. For more information about a specific command, consult the standards documentation for the protocol.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Command** method sends a command to the remote host and processes the result code sent back in response to that command. This method can be used to send custom commands to a server to take advantage of features or capabilities that may not be supported internally by the class library.

The ***isMultiLine*** parameter is used by the method to determine if multiple lines of data will be returned by the server as the result of a command. Unlike a single line response, which consists of a result code and result string, a multi-line response consists of one or more lines of text, terminated by a special end-of-data marker.

To determine the specific status code returned by the server, check the value of the **ResultCode** property after the method returns.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Command Overload List

NntpClient.Connect Method

Establish a connection with a remote host.

Overload List

Establish a connection with a remote host.

public bool Connect();

Establish a connection with a remote host.

public bool Connect(string);

Establish a connection with a remote host.

public bool Connect(string,int);

Establish a connection with a remote host.

public bool Connect(string,int,int);

Establish a connection with a remote host.

public bool Connect(string,int,int,NewsOptions);

Establish a connection with a remote host.

public bool Connect(string,int,string,string,int,NewsOptions);

See Also

NntpClient Class | SocketTools Namespace

NntpClient.Connect Method ()

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect() As Boolean
```

[C#]

```
public bool Connect();
```

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection using assigned property values.

The value of the **HostName** or **HostAddress** property will be used to determine the host name or address to connect to.

The value of the **RemotePort** property will be used to determine the port number to connect to.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Connect Overload List

NntpClient.Connect Method (String)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Connect Overload List

NntpClient.Connect Method (String, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Connect Overload List

NntpClient.Connect Method (String, Int32, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   int timeout
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Connect Overload List

NntpClient.Connect Method (String, Int32, Int32, NewsOptions)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer, _
   ByVal options As NewsOptions _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   int timeout,
   NewsOptions options
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

options

One or more of the NewsOptions enumeration flags.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Connect Overload List

NntpClient.Connect Method (String, Int32, String, String, Int32, NewsOptions)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal timeout As Integer, _
   ByVal options As NewsOptions _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   int timeout,
   NewsOptions options
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName

A string which specifies the username which will be used to authenticate the client session with the remote host. Not all news servers require the client to authenticate the session.

userPassword

A string which specifies the password which will be used to authenticate the client session with the remote host. Not all news servers require the client to authenticate the session.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

options

One or more of the NewsOptions enumeration flags.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns

false, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Connect Overload List

NntpClient.CreateArticle Method

Creates a new article in the current newsgroup.

[Visual Basic]

```
Public Function CreateArticle() As Boolean
```

[C#]

```
public bool CreateArticle();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateArticle** method sends the POST command to the news server. Not all servers permit clients to post articles. The client application is responsible for generating the message headers as well as the body of the message. News articles conform to the same general characteristics of an email message.

The **CloseArticle** method must be called once the contents of the article has been written to the server.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.Disconnect Method

Terminate the connection with a remote host.

[Visual Basic]

```
Public Sub Disconnect()
```

[C#]

```
public void Disconnect();
```

Remarks

The **Disconnect** method terminates the connection with the remote host and releases the client handle allocated by the class. Note that the socket is not immediately released when the connection is terminated and will enter a wait state for two minutes. After the time wait period has elapsed, the client will be released by the operating system. This is a normal safety mechanism to handle any packets that may arrive after the connection has been closed.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.Dispose Method

Releases all resources used by NntpClient.

Overload List

Releases all resources used by NntpClient.

public void Dispose();

Releases the unmanaged resources allocated by the NntpClient class and optionally releases the managed resources.

protected virtual void Dispose(bool);

See Also

NntpClient Class | SocketTools Namespace

NntpClient.Dispose Method ()

Releases all resources used by NntpClient.

[Visual Basic]

```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]

```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Dispose Overload List

NntpClient.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the NntpClient class and optionally releases the managed resources.

[Visual Basic]

```
Overridable Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]

```
protected virtual void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **NntpClient** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Dispose Overload List

NntpClient.Finalize Method

Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.

[Visual Basic]

```
Overrides Protected Sub Finalize()
```

[C#]

```
protected override void Finalize();
```

See Also

NntpClient Class | SocketTools Namespace

NntpClient.GetArticle Method

Retrieve an article from the server.

Overload List

Retrieve an article from the server.

public bool GetArticle(int,byte[],ref int);

Retrieve an article from the server.

public bool GetArticle(int,ref string);

Retrieve an article from the server.

public bool GetArticle(string,byte[],ref int);

Retrieve an article from the server.

public bool GetArticle(string,ref string);

See Also

NntpClient Class | SocketTools Namespace

NntpClient.GetArticle Method (Int32, Byte[], Int32)

Retrieve an article from the server.

[Visual Basic]

```
Overloads Public Function GetArticle( _
   ByVal articleId As Integer, _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool GetArticle(
   int articleId,
   byte[] buffer,
   ref int length
);
```

Parameters

articleId

An integer value which specifies the number of the article to retrieve from the server. This value must be greater than zero.

buffer

A byte array that the news article will be stored in.

length

An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetArticle** method is used to retrieve an article from the server and copy it into a local buffer. This method will cause the current thread to block until the article transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The first available article in the newsgroup can be determined by checking the value of the **FirstArticle** property. The last available article in the newsgroup is returned by the **LastArticle** property.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.GetArticle Overload List

NntpClient.GetArticle Method (String, Byte[], Int32)

Retrieve an article from the server.

```
[Visual Basic]
Overloads Public Function GetArticle( _
   ByVal messageId As String, _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

```
[C#]
public bool GetArticle(
   string messageId,
   byte[] buffer,
   ref int length
);
```

Parameters

messageId
A string value which specifies the message ID of the article to retrieve from the server.

buffer
A byte array that the news article will be stored in.

length
An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetArticle** method is used to retrieve an article from the server and copy it into a local buffer. This method will cause the current thread to block until the article transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.GetArticle Overload List

NntpClient.GetFirstArticle Method

Return information about all articles available in the current newsgroup.

Overload List

Return information about all articles available in the current newsgroup.

public bool GetFirstArticle(ref NewsArticle);

Return information about a selected range of articles in the current newsgroup.

public bool GetFirstArticle(int,int,ref NewsArticle);

See Also

NntpClient Class | SocketTools Namespace

NntpClient.GetFirstArticle Method (NewsArticle)

Return information about all articles available in the current newsgroup.

[Visual Basic]

```
Overloads Public Function GetFirstArticle( _
   ByRef article As NewsArticle _
) As Boolean
```

[C#]

```
public bool GetFirstArticle(
   ref NewsArticle article
);
```

Parameters

article

A NewsArticle structure which will contain information about the article when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFirstArticle** method returns information about the first article in the currently selected newsgroup. This method is used in conjunction with the **GetNextArticle** method to enumerate all of the articles in the newsgroup. Typically this is used to provide the user with a list of articles to access.

While the articles in the newsgroup are being listed, the client cannot retrieve the contents of a specific article. For example, the **GetArticle** method cannot be called while inside a loop calling **GetNextArticle**. The client should store those articles which it wants to retrieve in an array, and then once all of the articles have been listed, it can begin calling **GetArticle** for each article number to retrieve the article text.

The values of the **FirstArticle** and **LastArticle** properties determines the ranges of articles returned by this method.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.GetFirstArticle Overload List

NntpClient.GetFirstArticle Method (Int32, Int32, NewsArticle)

Return information about a selected range of articles in the current newsgroup.

[Visual Basic]

```
Overloads Public Function GetFirstArticle( _
   ByVal firstArticle As Integer, _
   ByVal lastArticle As Integer, _
   ByRef article As NewsArticle _
) As Boolean
```

[C#]

```
public bool GetFirstArticle(
   int firstArticle,
   int lastArticle,
   ref NewsArticle article
);
```

Parameters

firstArticle

An integer which specifies the first article to return. A value of -1 specifies that the first available article in the newsgroup should be returned.

lastArticle

An integer which specifies the last article to return. A value of -1 specifies that the last available article in the newsgroup should be returned.

article

A NewsArticle structure which will contain information about the article when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFirstArticle** method returns information about the first article in the currently selected newsgroup. This method is used in conjunction with the **GetNextArticle** method to enumerate all of the articles in the newsgroup. Typically this is used to provide the user with a list of articles to access.

While the articles in the newsgroup are being listed, the client cannot retrieve the contents of a specific article. For example, the **GetArticle** method cannot be called while inside a loop calling **GetNextArticle**. The client should store those articles which it wants to retrieve in an array, and then once all of the articles have been listed, it can begin calling **GetArticle** for each article number to retrieve the article text.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.GetFirstArticle Overload List

NntpClient.GetFirstGroup Method

Return information about all available newsgroups.

Overload List

Return information about all available newsgroups.

public bool GetFirstGroup(ref NewsGroup);

Return information about newsgroups created after the specified date.

public bool GetFirstGroup(DateTime,ref NewsGroup);

See Also

NntpClient Class | SocketTools Namespace

NntpClient.GetFirstGroup Method (NewsGroup)

Return information about all available newsgroups.

[Visual Basic]

```
Overloads Public Function GetFirstGroup( _
   ByRef group As NewsGroup _
) As Boolean
```

[C#]

```
public bool GetFirstGroup(
   ref NewsGroup group
);
```

Parameters

group

A NewsGroup structure which will contain information about the newsgroup when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFirstGroup** method returns information about the first newsgroup on the server. This method is used in conjunction with the **GetNextGroup** method to enumerate all of the available newsgroups. Typically this is used to provide the user with a list of newsgroups to select. If the **LastUpdate** property is set, then only newsgroups that have been created since that date will be returned.

While the the newsgroups are being listed, the client cannot select a newsgroup or retrieve the contents of a specific article. The client should store those newsgroups which it wants to retrieve articles from, and then once all of the newsgroups have been listed, it can then select each newsgroup and retrieve the available articles from that group.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.GetFirstGroup Overload List

NntpClient.GetFirstGroup Method (DateTime, NewsGroup)

Return information about newsgroups created after the specified date.

```
[Visual Basic]
Overloads Public Function GetFirstGroup( _
   ByVal lastUpdate As Date, _
   ByRef group As NewsGroup _
) As Boolean
```

```
[C#]
public bool GetFirstGroup(
   DateTime lastUpdate,
   ref NewsGroup group
);
```

Parameters

lastUpdate

A **System.DateTime** structure. Only those newsgroups which were created after this date and time will be returned. This is useful for checking for newsgroups that have been recently added to the server without incurring the overhead of listing every available newsgroup.

group

A NewsGroup structure which will contain information about the newsgroup when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetFirstGroup** method returns information about the first newsgroup on the server. This method is used in conjunction with the **GetNextGroup** method to enumerate all of the available newsgroups. Typically this is used to provide the user with a list of newsgroups to select.

While the the newsgroups are being listed, the client cannot select a newsgroup or retrieve the contents of a specific article. The client should store those newsgroups which it wants to retrieve articles from, and then once all of the newsgroups have been listed, it can then select each newsgroup and retrieve the available articles from that group.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.GetFirstGroup Overload List

NntpClient.GetHeaders Method

Retrieves the headers for the specified article from the server.

Overload List

Retrieves the headers for the specified article from the server.

public bool GetHeaders(int,byte[],ref int);

Retrieves the headers for the specified article from the server.

public bool GetHeaders(int,ref string);

Retrieves the headers for the specified article from the server.

public bool GetHeaders(string,byte[],ref int);

Retrieves the headers for the specified article from the server.

public bool GetHeaders(string,ref string);

See Also

NntpClient Class | SocketTools Namespace

NntpClient.GetHeaders Method (Int32, Byte[], Int32)

Retrieves the headers for the specified article from the server.

[Visual Basic]

```
Overloads Public Function GetHeaders( _
   ByVal articleId As Integer, _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool GetHeaders(
   int articleId,
   byte[] buffer,
   ref int length
);
```

Parameters

articleId
An integer value which specifies the article to retrieve from the server. This value must be greater than zero.

buffer
A byte array that will contain the article headers when the method returns.

length
An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeaders** method is used to retrieve an article header block from the server and copy it into a local buffer. This method will cause the current thread to block until the article transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The first available article in the newsgroup can be determined by checking the value of the **FirstArticle** property. The last available article in the newsgroup is returned by the **LastArticle** property.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.GetHeaders Overload List

NntpClient.GetHeaders Method (String, Byte[], Int32)

Retrieves the headers for the specified article from the server.

[Visual Basic]

```
Overloads Public Function GetHeaders( _
   ByVal messageId As String, _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool GetHeaders(
   string messageId,
   byte[] buffer,
   ref int length
);
```

Parameters

messageId
A string value which specifies the message number of the article to retrieve from the server.

buffer
A byte array that will contain the article headers when the method returns.

length
An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeaders** method is used to retrieve an article header block from the server and copy it into a local buffer. This method will cause the current thread to block until the article transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.GetHeaders Overload List

NntpClient.GetNextArticle Method

Return information about the next article available in the current newsgroup.

[Visual Basic]

```
Public Function GetNextArticle( _
   ByRef article As NewsArticle _
) As Boolean
```

[C#]

```
public bool GetNextArticle(
   ref NewsArticle article
);
```

Parameters

article

A NewsArticle structure which will contain information about the article when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetNextArticle** method returns information about the next available article in the currently selected newsgroup. This method is used in conjunction with the **GetFirstArticle** method to enumerate all of the articles in the newsgroup. Typically this is used to provide the user with a list of articles to access.

While the articles in the newsgroup are being listed, the client cannot retrieve the contents of a specific article. For example, the **GetArticle** method cannot be called while inside a loop calling **GetNextArticle**. The client should store those articles which it wants to retrieve in an array, and then once all of the articles have been listed, it can begin calling **GetArticle** for each article number to retrieve the article text.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.GetNextGroup Method

Return information about the next available newsgroup.

[Visual Basic]

```
Public Function GetNextGroup( _
   ByRef group As NewsGroup _
) As Boolean
```

[C#]

```
public bool GetNextGroup(
   ref NewsGroup group
);
```

Parameters

group
A NewsGroup structure which will contain information about the newsgroup when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetNextGroup** method returns information about the next available newsgroup on the server. This method is used in conjunction with the **GetFirstGroup** method to enumerate all of the available newsgroups. Typically this is used to provide the user with a list of newsgroups to select. If the **LastUpdate** property is set, then only newsgroups that have been created since that date will be returned.

While the the newsgroups are being listed, the client cannot select a newsgroup or retrieve the contents of a specific article. The client should store those newsgroups which it wants to retrieve articles from, and then once all of the newsgroups have been listed, it can then select each newsgroup and retrieve the available articles from that group.

A program should use either the **ListGroups** method or the **GetFirstGroup** and **GetNextGroup** methods, but never in combination with one another.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.Initialize Method

Initialize an instance of the NntpClient class.

Overload List

Initialize an instance of the NntpClient class.

```
public bool Initialize();
```

Initialize an instance of the NntpClient class.

```
public bool Initialize(string);
```

See Also

NntpClient Class | SocketTools Namespace | Uninitialize Method

NntpClient.Initialize Method ()

Initialize an instance of the NntpClient class.

[Visual Basic]

```
Overloads Public Function Initialize() As Boolean
```

[C#]

```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the NntpClient class, loading the networking library and allocating resources for the current thread. Typically it is not necessary to explicitly call this method because the instance of the class is initialized by the class constructor. However, if the **Uninitialize** method is called, the class must be re-initialized before any other methods are called.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Initialize Overload List | Uninitialize Method

NntpClient.Initialize Method (String)

Initialize an instance of the NntpClient class.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the NntpClient class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the NntpClient class.

Example

The following example shows how to use the Initialize method to initialize an instance of the class. This example assumes that the license key string has been defined in code.

```
SocketTools.NntpClient nntpClient = new SocketTools.NntpClient();

if (nntpClient.Initialize(strLicenseKey) == false)
{
    MessageBox.Show(nntpClient.LastErrorString, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    return;
}
```

```
Dim nntpClient As New SocketTools.NntpClient

If nntpClient.Initialize(strLicenseKey) = False Then
    MsgBox(nntpClient.LastErrorString, vbIconExclamation)
    Exit Sub
End If
```

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

NntpClient.OpenArticle Method

Opens the current article in the currently selected newsgroup.

Overload List

Opens the current article in the currently selected newsgroup.

public bool OpenArticle();

Opens the specified article in the currently selected newsgroup.

public bool OpenArticle(int);

See Also

NntpClient Class | SocketTools Namespace

NntpClient.OpenArticle Method ()

Opens the current article in the currently selected newsgroup.

[Visual Basic]

```
Overloads Public Function OpenArticle() As Boolean
```

[C#]

```
public bool OpenArticle();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **OpenArticle** method opens the specified article in the currently selected newsgroup. The article data can be read using the **Read** method, and once all of the data has been returned, the **CloseArticle** method should be used to close the article on the server.

The **Article** property returns the current article number.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.OpenArticle Overload List

NntpClient.OpenArticle Method (Int32)

Opens the specified article in the currently selected newsgroup.

[Visual Basic]

```
Overloads Public Function OpenArticle( _
   ByVal articleId As Integer _
) As Boolean
```

[C#]

```
public bool OpenArticle(
   int articleId
);
```

Parameters

articleId

An integer value which specifies the number of the article to retrieve from the server. This value must be greater than zero and it becomes the current article number for the selected newsgroup.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **OpenArticle** method opens the specified article in the currently selected newsgroup. The article data can be read using the **Read** method, and once all of the data has been returned, the **CloseArticle** method should be used to close the article on the server.

The first available article in the current newsgroup can be determined by checking the value of the **FirstArticle** property. The last available article in the current newsgroup is returned by the **LastArticle** property.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.OpenArticle Overload List

NntpClient.PostArticle Method

Post a new article to the currently selected newsgroup.

Overload List

Post a new article to the currently selected newsgroup.

public bool PostArticle(byte[],int);

Post a new article to the currently selected newsgroup.

public bool PostArticle(string);

See Also

NntpClient Class | SocketTools Namespace

NntpClient.PostArticle Method (Byte[], Int32)

Post a new article to the currently selected newsgroup.

[Visual Basic]

```
Overloads Public Function PostArticle( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool PostArticle(
   byte[] buffer,
   int length
);
```

Parameters

buffer
A byte array which contains the data to be posted to the server.

length
An integer value which specifies the number of bytes to write to the server. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PostArticle** method is used to submit the contents of the specified buffer to the server as a new article in the current newsgroup. This method will cause the current thread to block until the article transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Not all newsgroups permit new articles to be posted, and some newsgroups may require that you email the article to a moderator for approval instead of posting directly to the group. It may be required that the client authenticate itself using the **Authenticate** method prior to posting the article.

A news article is similar to an email message in that it contains one or more header fields, followed by an empty line, followed by the body of the article. Each line of text should be terminated by a carriage return/linefeed sequence of characters. The **SocketTools.MailMessage** class can be used to compose the news article if needed. Note that the article header must contain a header field named "Newsgroups" with a value that specifies the newsgroup or newsgroups the article is being posted to. If this header field is missing, the news server will reject the article.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.PostArticle Overload List

NntpClient.PostArticle Method (String)

Post a new article to the currently selected newsgroup.

[Visual Basic]

```
Overloads Public Function PostArticle( _
   ByVal buffer As String _
) As Boolean
```

[C#]

```
public bool PostArticle(
   string buffer
);
```

Parameters

buffer
A string which contains the data to be posted to the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **PostArticle** method is used to submit the contents of the specified buffer to the server as a new article in the current newsgroup. This method will cause the current thread to block until the article transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Not all newsgroups permit new articles to be posted, and some newsgroups may require that you email the article to a moderator for approval instead of posting directly to the group. It may be required that the client authenticate itself using the **Authenticate** method prior to posting the article.

A news article is similar to an email message in that it contains one or more header fields, followed by an empty line, followed by the body of the article. Each line of text should be terminated by a carriage return/linefeed sequence of characters. The **SocketTools.MailMessage** class can be used to compose the news article if needed. Note that the article header must contain a header field named "Newsgroups" with a value that specifies the newsgroup or newsgroups the article is being posted to. If this header field is missing, the news server will reject the article.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.PostArticle Overload List

NntpClient.Read Method

Read data from the server and store it in a byte array.

Overload List

Read data from the server and store it in a byte array.

public int Read(byte[]);

Read data from the server and store it in a byte array.

public int Read(byte[],int);

Read data from the server and store it in a string.

public int Read(ref string);

Read data from the server and store it in a string.

public int Read(ref string,int);

See Also

NntpClient Class | SocketTools Namespace

NntpClient.Read Method (Byte[])

Read data from the server and store it in a byte array.

[Visual Basic]

```
Overloads Public Function Read( _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Read(
   byte[] buffer
);
```

Parameters

buffer

A byte array that the data will be stored in.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the size of the byte array passed to the method. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Read Overload List

NntpClient.Read Method (Byte[], Int32)

Read data from the server and store it in a byte array.

[Visual Basic]

```
Overloads Public Function Read( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Read(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array that the data will be stored in.

length

An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the number of bytes specified. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Read Overload List

NntpClient.Read Method (String)

Read data from the server and store it in a string.

[Visual Basic]

```
Overloads Public Function Read( _
   ByRef buffer As String _
) As Integer
```

[C#]

```
public int Read(
   ref string buffer
);
```

Parameters

buffer

A string that will contain the data read from the client.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to a maximum of 4096 bytes. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Read Overload List

NntpClient.Read Method (String, Int32)

Read data from the server and store it in a string.

[Visual Basic]

```
Overloads Public Function Read( _
   ByRef buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Read(
   ref string buffer,
   int length
);
```

Parameters

buffer

A string that will contain the data read from the client.

length

An integer value which specifies the maximum number of bytes of data to read.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the number of bytes specified. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Read Overload List

NntpClient.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]
```
Public Sub Reset()
```

[C#]
```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. If a socket has been allocated, it will be released and any active connections will be terminated. All properties will be reset to their default values.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.SelectGroup Method

Selects the specified newsgroup as the current newsgroup.

[Visual Basic]

```
Public Function SelectGroup( _
   ByVal groupName As String _
) As Boolean
```

[C#]

```
public bool SelectGroup(
   string groupName
);
```

Parameters

groupName

A string which specifies the name of the newsgroup to select.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SelectGroup** method selects a newsgroup and updates the control with information about the group.

The values of the **FirstArticle** and **LastArticle** properties will be updated to reflect the range of available articles in the newsgroup. The value of the **Article** property will be set to the first available article in the newsgroup.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.StoreArticle Method

Retrieve an article from the selected newsgroup and store it in a file.

Overload List

Retrieve an article from the selected newsgroup and store it in a file.

public bool StoreArticle(int,string);

Retrieve the current article from the selected newsgroup and store it in a file.

public bool StoreArticle(string);

See Also

NntpClient Class | SocketTools Namespace

NntpClient.StoreArticle Method (Int32, String)

Retrieve an article from the selected newsgroup and store it in a file.

[Visual Basic]

```
Overloads Public Function StoreArticle( _
   ByVal articleId As Integer, _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool StoreArticle(
   int articleId,
   string fileName
);
```

Parameters

articleId

An integer value which specifies the number of the article to retrieve from the server. This value must be greater than zero.

fileName

A string which specifies the name of the file that will contain the article. If the file does not exist, it will be created. If the file already exists, it will be overwritten.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **StoreArticle** method retrieves an article from the server and stores it in a file on the local system. The contents of the article is stored as a text file, using the specified file name. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The first available article in the current newsgroup can be determined by checking the value of the **FirstArticle** property. The last available article in the current newsgroup is returned by the **LastArticle** property.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.StoreArticle Overload List

NntpClient.StoreArticle Method (String)

Retrieve the current article from the selected newsgroup and store it in a file.

[Visual Basic]
```
Overloads Public Function StoreArticle( _
   ByVal fileName As String _
) As Boolean
```

[C#]
```
public bool StoreArticle(
   string fileName
);
```

Parameters

fileName
A string which specifies the name of the file that will contain the article. If the file does not exist, it will be created. If the file already exists, it will be overwritten.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **StoreArticle** method retrieves an article from the server and stores it in a file on the local system. The contents of the article is stored as a text file, using the specified file name. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The **Article** property returns the current article number.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.StoreArticle Overload List

NntpClient.Uninitialize Method

Uninitialize the class library and release any resources allocated for the current thread.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method terminates any active connection, releases resources allocated for the current thread and unloads the networking library. After this method has been called, no further client operations may be performed until the class instance has been re-initialized.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the class is no longer needed.

See Also

NntpClient Class | SocketTools Namespace | Initialize Method

NntpClient.Write Method

Write one or more bytes of data to the server.

Overload List

Write one or more bytes of data to the server.

public int Write(byte[]);

Write one or more bytes of data to the server.

public int Write(byte[],int);

Write a string of characters to the server.

public int Write(string);

Write a string of characters to the server.

public int Write(string,int);

See Also

NntpClient Class | SocketTools Namespace

NntpClient.Write Method (Byte[])

Write one or more bytes of data to the server.

```
[Visual Basic]
Overloads Public Function Write( _
   ByVal buffer As Byte() _
) As Integer
```

```
[C#]
public int Write(
   byte[] buffer
);
```

Parameters

buffer
A byte array that contains the data to be written to the server.

Return Value

An integer value which specifies the number of bytes actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Write Overload List

NntpClient.Write Method (Byte[], Int32)

Write one or more bytes of data to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Write(
   byte[] buffer,
   int length
);
```

Parameters

buffer
A byte array that contains the data to be written to the server.

length
An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Write Overload List

NntpClient.Write Method (String)

Write a string of characters to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String _
) As Integer
```

[C#]

```
public int Write(
   string buffer
);
```

Parameters

buffer

A string which contains the data to be written to the server.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Write Overload List

NntpClient.Write Method (String, Int32)

Write a string of characters to the server.

[Visual Basic]
```
Overloads Public Function Write( _
   ByVal buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]
```
public int Write(
   string buffer,
   int length
);
```

Parameters

buffer
A string which contains the data to be written to the server.

length
An integer value which specifies the maximum number of characters to write. This value cannot be larger than the length of the string specified by the caller.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

NntpClient Class | SocketTools Namespace | NntpClient.Write Overload List

NntpClient Events

The events of the **NntpClient** class are listed below. For a complete list of **NntpClient** class members, see the NntpClient Members topic.

Public Instance Events

⚡ OnCancel	Occurs when a blocking client operation is canceled.
⚡ OnCommand	Occurs when the client sends a command to the remote host and receives a reply indicating the result of that command.
⚡ OnConnect	Occurs when a connection is established with the remote host.
⚡ OnDisconnect	Occurs when the remote host disconnects from the local system.
⚡ OnError	Occurs when an client operation fails.
⚡ OnProgress	Occurs as a data stream is being read or written to the client.
⚡ OnRead	Occurs when data is available to be read from the client.
⚡ OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
⚡ OnWrite	Occurs when data can be written to the client.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.OnCancel Event

Occurs when a blocking client operation is canceled.

[Visual Basic]

```
Public Event OnCancel As EventHandler
```

[C#]

```
public event EventHandler OnCancel;
```

Remarks

The **OnCancel** event is generated when a blocking client operation, such as sending or receiving data, is canceled with the **Cancel** method. To assist in determining which operation was canceled, check the value of the **Status** property.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.OnCommand Event

Occurs when the client sends a command to the remote host and receives a reply indicating the result of that command.

[Visual Basic]
```
Public Event OnCommand As OnCommandEventHandler
```

[C#]
```
public event OnCommandEventHandler OnCommand;
```

Event Data

The event handler receives an argument of type NntpClient.CommandEventArgs containing data related to this event. The following **NntpClient.CommandEventArgs** properties provide information specific to this event.

Property	Description
ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.

Remarks

The **OnCommand** event is generated when the client receives a reply from the server after some action has been taken.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.CommandEventArgs Class

Provides data for the OnCommand event.

For a list of all members of this type, see NntpClient.CommandEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.NntpClient.CommandEventArgs

[Visual Basic]

```
Public Class NntpClient.CommandEventArgs
    Inherits EventArgs
```

[C#]

```
public class NntpClient.CommandEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

CommandEventArgs specifies the result code and result string for the last command executed by the server.

The OnCommand event occurs whenever a command is executed on the server.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NntpClient (in SocketTools.NntpClient.dll)

See Also

NntpClient.CommandEventArgs Members | SocketTools Namespace

NntpClient.CommandEventArgs Members

NntpClient.CommandEventArgs overview

Public Instance Constructors

NntpClient.CommandEventArgs Constructor	Initializes a new instance of the NntpClient.CommandEventArgs class.

Public Instance Properties

ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

NntpClient.CommandEventArgs Class | SocketTools Namespace

NntpClient.CommandEventArgs Constructor

Initializes a new instance of the NntpClient.CommandEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public NntpClient.CommandEventArgs();
```

See Also

NntpClient.CommandEventArgs Class | SocketTools Namespace

NntpClient.CommandEventArgs Properties

The properties of the **NntpClient.CommandEventArgs** class are listed below. For a complete list of **NntpClient.CommandEventArgs** class members, see the NntpClient.CommandEventArgs Members topic.

Public Instance Properties

ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.

See Also

NntpClient.CommandEventArgs Class | SocketTools Namespace

NntpClient.CommandEventArgs.ResultCode Property

Gets a value which specifies the last result code returned by the server.

[Visual Basic]
```
Public ReadOnly Property ResultCode As Integer
```

[C#]
```
public int ResultCode {get;}
```

Property Value

An integer value which specifies the last result code returned by the server.

Remarks

This property should be checked after the **Command** method is used to execute a command on the server to determine if the operation was successful. Result codes are three-digit numeric values returned by the remote server and may be broken down into the following ranges:

ResultCode	Description
100-199	Positive preliminary result. This indicates that the requested action is being initiated, and the client should expect another reply from the server before proceeding.
200-299	Positive completion result. This indicates that the server has successfully completed the requested action.
300-399	Positive intermediate result. This indicates that the requested action cannot complete until additional information is provided to the server.
400-499	Transient negative completion result. This indicates that the requested action did not take place, but the error condition is temporary and may be attempted again.
500-599	Permanent negative completion result. This indicates that the requested action did not take place.

It is important to note that while some result codes have become standardized, not all servers respond to commands using the same result codes. For example, one server may respond with a result code of 221 to indicate success, while another may respond with a value of 235. It is recommended that applications check for ranges of values to determine if a command was successful, not a specific value.

See Also

NntpClient.CommandEventArgs Class | SocketTools Namespace

NntpClient.CommandEventArgs.ResultString Property

Gets a string value which describes the result of the previous command.

[Visual Basic]
```
Public ReadOnly Property ResultString As String
```

[C#]
```
public string ResultString {get;}
```

Property Value

A string which describes the result of the previous command executed on the server.

Remarks

This string is generated by the remote server, and typically is used to describe the result code. For example, if an error is indicated by the result code, the result string may describe the condition that caused the error.

See Also

NntpClient.CommandEventArgs Class | SocketTools Namespace

NntpClient.OnConnect Event

Occurs when a connection is established with the remote host.

[Visual Basic]

```
Public Event OnConnect As EventHandler
```

[C#]

```
public event EventHandler OnConnect;
```

Remarks

The **OnConnect** event occurs when a connection is made with a remote host as a result of a **Connect** method call. When the **Connect** method is called and the **Blocking** property is set to **false**, a socket is created but the connection is not actually established until after this event occurs. Between the time connection process is started and this event fires, no operation may be performed on the client other than calling the **Disconnect** method.

This event is only generated if the client is in non-blocking mode.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.OnDisconnect Event

Occurs when the remote host disconnects from the local system.

[Visual Basic]

```
Public Event OnDisconnect As EventHandler
```

[C#]

```
public event EventHandler OnDisconnect;
```

Remarks

The **OnDisconnect** event occurs when the remote host closes its connection, terminating the client session with the application. Because there may still be data in the client receive buffers, you should continue to read data from the client until the **Read** method returns a value of 0. Once all of the data has been read, you should call the **Disconnect** method to close the local socket and release the resources allocated for the client.

This event is only generated if the client is in non-blocking mode.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.OnError Event

Occurs when an client operation fails.

[Visual Basic]

```
Public Event OnError As OnErrorEventHandler
```

[C#]

```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type NntpClient.ErrorEventArgs containing data related to this event. The following **NntpClient.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a client operation fails.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see NntpClient.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.NntpClient.ErrorEventArgs

```
[Visual Basic]
Public Class NntpClient.ErrorEventArgs
    Inherits EventArgs
```

```
[C#]
public class NntpClient.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NntpClient (in SocketTools.NntpClient.dll)

See Also

NntpClient.ErrorEventArgs Members | SocketTools Namespace

NntpClient.ErrorEventArgs Members

NntpClient.ErrorEventArgs overview

Public Instance Constructors

NntpClient.ErrorEventArgs Constructor	Initializes a new instance of the NntpClient.ErrorEventArgs class.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

NntpClient.ErrorEventArgs Class | SocketTools Namespace

NntpClient.ErrorEventArgs Constructor

Initializes a new instance of the NntpClient.ErrorEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public NntpClient.ErrorEventArgs();
```

See Also

NntpClient.ErrorEventArgs Class | SocketTools Namespace

NntpClient.ErrorEventArgs Properties

The properties of the **NntpClient.ErrorEventArgs** class are listed below. For a complete list of **NntpClient.ErrorEventArgs** class members, see the NntpClient.ErrorEventArgs Members topic.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

NntpClient.ErrorEventArgs Class | SocketTools Namespace

NntpClient.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

NntpClient.ErrorEventArgs Class | SocketTools Namespace | Error Property

NntpClient.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public NntpClient.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

NntpClient.ErrorEventArgs Class | SocketTools Namespace | Description Property

NntpClient.OnProgress Event

Occurs as a data stream is being read or written to the client.

[Visual Basic]

```
Public Event OnProgress As OnProgressEventHandler
```

[C#]

```
public event OnProgressEventHandler OnProgress;
```

Event Data

The event handler receives an argument of type NntpClient.ProgressEventArgs containing data related to this event. The following **NntpClient.ProgressEventArgs** properties provide information specific to this event.

Property	Description
Article	Gets the current article number.
BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
BytesTotal	Gets a value which specifies the total number of bytes in the data stream.
Percent	Gets a value which specifies the percentage of data that has been read or written.

Remarks

The **OnProgress** event occurs as a data stream is being read or written to the client. If large amounts of data are being read or written, this event can be used to update a progress bar or other user-interface component to provide the user with some visual feedback on the progress of the operation.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.ProgressEventArgs Class

Provides data for the OnProgress event.

For a list of all members of this type, see NntpClient.ProgressEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.NntpClient.ProgressEventArgs

[Visual Basic]

```
Public Class NntpClient.ProgressEventArgs
    Inherits EventArgs
```

[C#]

```
public class NntpClient.ProgressEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ProgressEventArgs specifies the number of bytes copied from the data stream, the total number of bytes in the data stream and a completion percentage.

The OnProgress event occurs as a data stream is being read or written to the client.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NntpClient (in SocketTools.NntpClient.dll)

See Also

NntpClient.ProgressEventArgs Members | SocketTools Namespace

NntpClient.ProgressEventArgs Members

NntpClient.ProgressEventArgs overview

Public Instance Constructors

NntpClient.ProgressEventArgs Constructor	Initializes a new instance of the NntpClient.ProgressEventArgs class.

Public Instance Properties

Article	Gets the current article number.
BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
BytesTotal	Gets a value which specifies the total number of bytes in the data stream.
Percent	Gets a value which specifies the percentage of data that has been read or written.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

NntpClient.ProgressEventArgs Class | SocketTools Namespace

NntpClient.ProgressEventArgs Constructor

Initializes a new instance of the NntpClient.ProgressEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public NntpClient.ProgressEventArgs();
```

See Also

NntpClient.ProgressEventArgs Class | SocketTools Namespace

NntpClient.ProgressEventArgs Properties

The properties of the **NntpClient.ProgressEventArgs** class are listed below. For a complete list of **NntpClient.ProgressEventArgs** class members, see the NntpClient.ProgressEventArgs Members topic.

Public Instance Properties

Article	Gets the current article number.
BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
BytesTotal	Gets a value which specifies the total number of bytes in the data stream.
Percent	Gets a value which specifies the percentage of data that has been read or written.

See Also

NntpClient.ProgressEventArgs Class | SocketTools Namespace

NntpClient.ProgressEventArgs.Article Property

Gets the current article number.

[Visual Basic]

```
Public ReadOnly Property Article As Integer
```

[C#]

```
public int Article {get;}
```

Property Value

An integer value which specifies the article number.

Remarks

The **Article** property specifies the article number that is currently being retrieved from the news server. If the **OnProgress** event occurs while a new article is being posted to the server, this property will return a value of zero.

See Also

NntpClient.ProgressEventArgs Class | SocketTools Namespace

NntpClient.ProgressEventArgs.BytesCopied Property

Gets a value which specifies the number of bytes of data that has been read or written.

[Visual Basic]

```
Public ReadOnly Property BytesCopied As Integer
```

[C#]

```
public int BytesCopied {get;}
```

Property Value

An integer value which specifies the number of bytes of data.

Remarks

The **BytesCopied** property specifies the number of bytes that have been read from the client and stored in the local stream buffer, or written from the stream buffer to the client.

See Also

NntpClient.ProgressEventArgs Class | SocketTools Namespace | BytesTotal Property | Percent Property

NntpClient.ProgressEventArgs.BytesTotal Property

Gets a value which specifies the total number of bytes in the data stream.

[Visual Basic]

```
Public ReadOnly Property BytesTotal As Integer
```

[C#]

```
public int BytesTotal {get;}
```

Property Value

An integer value which specifies the number of bytes of data.

Remarks

The **BytesTotal** property specifies the total amount of data being read from the client and stored in the data stream, or written from the data stream to the client. If the amount of data was unknown or unspecified at the time the method call was made, then this value will always be the same as the **BytesCopied** property.

See Also

NntpClient.ProgressEventArgs Class | SocketTools Namespace | BytesCopied Property | Percent Property

NntpClient.ProgressEventArgs.Percent Property

Gets a value which specifies the percentage of data that has been read or written.

[Visual Basic]

```
Public ReadOnly Property Percent As Integer
```

[C#]

```
public int Percent {get;}
```

Property Value

An integer value which specifies a percentage.

Remarks

The **Percent** property specifies the percentage of data that has been transmitted, expressed as an integer value between 0 and 100, inclusive. If the maximum size of the data stream was not specified by the caller, this value will always be 100.

See Also

NntpClient.ProgressEventArgs Class | SocketTools Namespace | BytesCopied Property | BytesTotal Property

NntpClient.OnRead Event

Occurs when data is available to be read from the client.

[Visual Basic]

```
Public Event OnRead As EventHandler
```

[C#]

```
public event EventHandler OnRead;
```

Remarks

The **OnRead** event occurs when data is available to be read from the client. This event is level-triggered, which means that once this event fires, it will not occur again until some data has been read from the client. This design prevents an application from being flooded with event notifications. It is recommended that your application read all of the available data from the server and store it in a local buffer for processing. See the example below.

This event is only generated if the client is in non-blocking mode.

Example

```
Private Sub Socket_OnRead(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Socket.OnRead
    Dim strBuffer As String
    Dim nRead As Integer

    Do
        ' Read up to m_nBufferSize bytes of data from the server
        nRead = Socket.Read(strBuffer, m_nBufferSize)

        If nRead > 0 Then
            ' Append the data to an internal buffer for processing
            m_dataBuffer = m_dataBuffer + strBuffer
        End If
    Loop Until nRead < 1

    ProcessData()
End Sub
```

See Also

NntpClient Class | SocketTools Namespace

NntpClient.OnTimeout Event

Occurs when a blocking operation fails to complete before the timeout period elapses.

[Visual Basic]

```
Public Event OnTimeout As EventHandler
```

[C#]

```
public event EventHandler OnTimeout;
```

Remarks

The **OnTimeout** event occurs when a blocking operation, such as sending or receiving data on the client, fails to complete before the specified timeout period elapses. The timeout period for a blocking operation can be adjusted by setting the **Timeout** property.

This event is only generated if the client is in blocking mode.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.OnWrite Event

Occurs when data can be written to the client.

[Visual Basic]

```
Public Event OnWrite As EventHandler
```

[C#]

```
public event EventHandler OnWrite;
```

Remarks

The **OnWrite** event occurs when the application can write data to the client. This event will typically occur when a connection is first established with the remote host, and after the **Write** method has failed because there was insufficient memory available in the client send buffers. In the second case, when some of the buffered data has been successfully sent to the remote host and there is space available in the send buffers, this event is used to signal the application that it may attempt to send more data.

This event is only generated if the client is in non-blocking mode.

See Also

NntpClient Class | SocketTools Namespace

NntpClient.ErrorCode Enumeration

Specifies the error codes returned by the NntpClient class.

[Visual Basic]

```
Public Enum NntpClient.ErrorCode
```

[C#]

```
public enum NntpClient.ErrorCode
```

Remarks

The NntpClient class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.
errorProxyAuthenticationRequired	Proxy authentication required.

errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.
errorUnsupportedMediaType	Request specified an unsupported media type.

errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.
errorNoMoreMailboxes	No more mailboxes exist on this server.

errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.
errorInvalidThreadHandle	The specified thread handle is no longer valid.

errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been defined.

errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or unrecognized.

errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NntpClient (in SocketTools.NntpClient.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

NntpClient.NewsGroupAccess Enumeration

Specifies the newsgroup access modes that the NntpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum NntpClient.NewsGroupAccess
```

[C#]
```
[Flags]
public enum NntpClient.NewsGroupAccess
```

Members

Member Name	Description	Value
groupReadOnly	The group is read-only and cannot be modified. Attempts to post articles to the newsgroup will result in an error.	0
groupReadWrite	Articles can be posted to the newsgroup. Even though a newsgroup is read-write, it may require that the client authenticate before being given permission to post articles to the server.	1
groupModerated	The newsgroup is moderated and articles can only be posted by the group moderator. To request that an article be posted to the newsgroup, you must email the message to the moderator.	2

Requirements

Namespace: SocketTools

Assembly: SocketTools.NntpClient (in SocketTools.NntpClient.dll)

See Also

SocketTools Namespace

NntpClient.NewsOptions Enumeration

Specifies the options that the NntpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum NntpClient.NewsOptions
```

[C#]

```
[Flags]
public enum NntpClient.NewsOptions
```

Remarks

The NntpClient class uses the **NewsOptions** enumeration to specify one or more options to be used when establishing a connection with a remote host. Multiple options may be specified if necessary.

Members

Member Name	Description	Value
optionNone	No option specified.	0
optionDefault	The default connection option. This is the same as specifying **optionNone**.	0
optionTunnel	This option specifies that a tunneled TCP connection and/or port-forwarding is being used to establish the connection to the server. This changes the behavior of the client with regards to internal checks of the destination IP address and remote port number, default capability selection and how the connection is established. This option also forces all connections to be outbound and enables the firewall compatibility features in the client.	1024
optionTrustedSite	This option specifies the server is trusted. The server certificate will not be validated and the connection will always be permitted. This option only affects connections using either the SSL or TLS protocols.	2048
optionSecure	This option specifies the client should attempt to establish a secure connection with the server. Note that the server must support secure connections using either the SSL or TLS protocol.	4096
optionSecureFallback	This option specifies the client should permit the use of less secure cipher	32768

	suites for compatibility with legacy servers. If this option is specified, the client will permit connections using TLS 1.0 and cipher suites that use RC4, MD5 and SHA1.	
optionPreferIPv6	This option specifies the client should prefer the use of IPv6 if the server hostname can be resolved to both an IPv6 and IPv4 address. This option is ignored if the local system does not have IPv6 enabled, or when the hostname can only be resolved to an IPv4 address. If the server hostname can only be resolved to an IPv6 address, the client will attempt to establish a connection using IPv6 regardless if this option has been specified.	262144
optionFreeThread	This option specifies that class methods may be called from any thread, and not only the thread that established the connection. Using this option disables certain internal safety checks that are made by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads.	524288

Requirements

Namespace: SocketTools

Assembly: SocketTools.NntpClient (in SocketTools.NntpClient.dll)

See Also

SocketTools Namespace

NntpClient.NewsStatus Enumeration

Specifies the status values that may be returned by the NntpClient class.

[Visual Basic]
```
Public Enum NntpClient.NewsStatus
```

[C#]
```
public enum NntpClient.NewsStatus
```

Remarks

The NntpClient class uses the **NewsStatus** enumeration to identify the current status of the client.

Members

Member Name	Description
statusUnused	A client session has not been created. Attempts to perform any network operations, such as sending or receiving data, will generate an error.
statusIdle	A client session has been created, but is not currently in use. A blocking socket operation can be executed at this point.
statusConnect	The client is in the process of establishing a connection with a remote host.
statusRead	The client is in the process of receiving data from a remote host.
statusWrite	The client is in the process of sending data to a remote host.
statusDisconnect	The client session is being closed and subsequent attempts to access the client will result in an error.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NntpClient (in SocketTools.NntpClient.dll)

See Also

SocketTools Namespace

NntpClient.SecureCipherAlgorithm Enumeration

Specifies the encryption algorithms that the NntpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum NntpClient.SecureCipherAlgorithm
```

[C#]
```
[Flags]
public enum NntpClient.SecureCipherAlgorithm
```

Remarks

The NntpClient class uses the **SecureCipherAlgorithm** enumeration to identify which encryption algorithm was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
cipherNone	No cipher has been selected. A secure connection has not been established with the remote host.	0
cipherRC2	The RC2 block cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.	1
cipherRC4	The RC4 stream cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.	2
cipherRC5	The RC5 block cipher was selected. This is a variable key length cipher which supports keys up to 2040 bits, in 8-bit increments.	4
cipherDES	The DES (Data Encryption Standard) block cipher was selected. This is a fixed key length cipher using 56-bit keys.	8
cipherDES3	The Triple DES block cipher was selected. This cipher encrypts the data three times using different keys, effectively using a 168-bit key length.	16
cipherDESX	A variant of the DES block cipher which XORs an extra 64-bits of the key before and after the plaintext has been encrypted, increasing the key size to 184 bits.	32
cipherAES	The Advanced Encryption Standard	64

	cipher (also known as the Rijndael cipher) is a fixed block size cipher which use a key size of 128, 192 or 256 bits. This cipher is supported on Windows XP SP3 SP3 and later versions of the operating system.	
cipherSkipjack	The Skipjack block cipher was selected. This is a fixed key length cipher, using 80-bit keys.	128

Requirements

Namespace: SocketTools

Assembly: SocketTools.NntpClient (in SocketTools.NntpClient.dll)

See Also

SocketTools Namespace

NntpClient.SecureHashAlgorithm Enumeration

Specifies the hash algorithms that the NntpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum NntpClient.SecureHashAlgorithm
```

[C#]

```
[Flags]
public enum NntpClient.SecureHashAlgorithm
```

Remarks

The NntpClient class uses the **SecureHashAlgorithm** enumeration to identify the message digest (hash) algorithm that was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
hashNone	No hash algorithm has been selected. This is not a secure connection with the server.	0
hashMD5	The MD5 algorithm was selected. This algorithm produces a 128-bit message digest. This algorithm is no longer considered to be cryptographically secure.	1
hashSHA	The SHA-1 algorithm was selected. This algorithm produces a 160-bit message digest. This algorithm is no longer considered to be cryptographically secure.	2
hashSHA256	The SHA-256 algorithm was selected. This algorithm produces a 256-bit message digest.	4
hashSHA384	The SHA-384 algorithm was selected. This algorithm produces a 384-bit message digest.	8
hashSHA512	The SHA-512 algorithm was selected. This algorithm produces a 512-bit message digest.	16

Requirements

Namespace: SocketTools

Assembly: SocketTools.NntpClient (in SocketTools.NntpClient.dll)

See Also

SocketTools Namespace

NntpClient.SecureKeyAlgorithm Enumeration

Specifies the key exchange algorithms that the NntpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum NntpClient.SecureKeyAlgorithm
```

[C#]

```
[Flags]
public enum NntpClient.SecureKeyAlgorithm
```

Remarks

The NntpClient class uses the **SecureKeyAlgorithm** enumeration to identify the key exchange algorithm that was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
keyExchangeNone	No key exchange algorithm has been selected. This is not a secure connection with the server.	0
keyExchangeRSA	The RSA public key exchange algorithm has been selected.	1
keyExchangeKEA	The KEA public key exchange algorithm has been selected. This is an improved version of the Diffie-Hellman public key algorithm.	2
keyExchangeDH	The Diffie-Hellman public key exchange algorithm has been selected.	4
keyExchangeECDH	The Elliptic Curve Diffie-Hellman key exchange algorithm was selected. This is a variant of the Diffie-Hellman algorithm which uses elliptic curve cryptography. This key exchange algorithm is only supported on Windows XP SP3 SP3 and later versions of the operating system.	8

Requirements

Namespace: SocketTools

Assembly: SocketTools.NntpClient (in SocketTools.NntpClient.dll)

See Also

SocketTools Namespace

NntpClient.SecurityCertificate Enumeration

Specifies the security certificate status values that may be returned by the NntpClient class.

[Visual Basic]
```
Public Enum NntpClient.SecurityCertificate
```

[C#]
```
public enum NntpClient.SecurityCertificate
```

Remarks

The NntpClient class uses the **SecurityCertificate** enumeration to identify the current status of the certificate that was provided by the remote host when a secure connection was established.

Members

Member Name	Description
certificateNone	No certificate information is available. A secure connection was not established with the server.
certificateValid	The certificate is valid.
certificateNoMatch	The certificate is valid, however the domain name specified in the certificate does not match the domain name of the remote host. The application can examine the **CertificateSubject** property to determine the site the certificate was issued to.
certificateExpired	The certificate has expired and is no longer valid. The application can examine the **CertificateExpires** property to determine when the certificate expired.
certificateRevoked	The certificate has been revoked and is no longer valid. It is recommended that the application immediately terminate the connection if this status is returned.
certificateUntrusted	The certificate has not been issued by a trusted authority, or the certificate is not trusted on the local host. It is recommended that the application immediately terminate the connection if this status is returned.
certificateInvalid	The certificate is invalid. This typically indicates that the internal structure of the certificate is damaged. It is recommended that the application immediately terminate the connection if this status is returned.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NntpClient (in SocketTools.NntpClient.dll)

See Also

SocketTools Namespace

NntpClient.SecurityProtocols Enumeration

Specifies the security protocols that the NntpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum NntpClient.SecurityProtocols
```

```
[C#]
[Flags]
public enum NntpClient.SecurityProtocols
```

Remarks

The NntpClient class uses the **SecurityProtocols** enumeration to specify one or more security protocols to be used when establishing a connection with a remote host. Multiple protocols may be specified if necessary and the actual protocol used will be negotiated with the remote host. It is recommended that most applications use **protocolDefault** when creating a secure connection.

Members

Member Name	Description	Value
protocolNone	No security protocol will be used, a secure connection will not be established.	0
protocolSSL2	The SSL 2.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections.	1
protocolSSL3	The SSL 3.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections. In most cases, this protocol is only selected if TLS is not supported by the server.	2
protocolTLS10	The TLS 1.0 protocol has been selected. This version of the protocol is commonly used by older servers and is the only version of TLS supported on Windows platforms prior to Windows 7 SP1 and Windows Server 2008 R2.	4
protocolTLS11	The TLS 1.1 protocol has been selected. This version of TLS is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of the	8

	operating system.	
protocolTLS12	The TLS 1.2 protocol has been selected. This is the default version of the protocol and is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of Windows. It is recommended that you use this version of TLS.	16
protocolSSL	Any version of the Secure Sockets Layer (SSL) protocol should be used. The actual protocol version used will be negotiated with the remote host.	3
protocolTLS	Any version of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with with preference for TLS 1.2.	28
protocolTLS1	Version 1.0, 1.1 or 1.2 of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with preference for TLS 1.2.	28
protocolDefault	The default selection of security protocols will be used when establishing a connection. The TLS 1.2, 1.1 and 1.0 protocols will be negotiated with the host, in that order of preference. This option will always request the latest version of the preferred security protocols and is the recommended value.	16
protocolUnknown	An unknown or unsupported security protocol has been specified. This value indicates an error condition.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.NntpClient (in SocketTools.NntpClient.dll)

See Also

SocketTools Namespace

NntpClient.TraceOptions Enumeration

Specifies the logging options that the NntpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum NntpClient.TraceOptions
```

[C#]

```
[Flags]
public enum NntpClient.TraceOptions
```

Remarks

The NntpClient class uses the **TraceOptions** enumeration to specify what kind of debugging information is written to the trace logfile. These options are only meaningful when trace logging is enabled by setting the **Trace** property to **true**.

Members

Member Name	Description	Value
traceDefault	The default trace logging option. This is the same as specifying the **traceInfo** option.	0
traceInfo	All network function calls are written to the trace file. This is the default value.	0
traceError	Only those network function calls which fail are recorded in the trace file.	1
traceWarning	Only those network function calls which fail, or return values which indicate a warning, are recorded in the trace file.	2
traceHexDump	All network function calls are written to the trace file, plus all the data that is sent or received is displayed, in both ASCII and hexadecimal format.	4
traceProcess	All function calls in the current process are logged, rather than only those functions in the current thread. This option is useful for multithreaded applications that are using worker threads.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.NntpClient (in SocketTools.NntpClient.dll)

See Also

SocketTools Namespace

NntpClient.OnCommandEventHandler Delegate

Represents the method that will handle the OnCommand event.

[Visual Basic]

```
Public Delegate Sub NntpClient.OnCommandEventHandler( _
   ByVal sender As Object, _
   ByVal e As CommandEventArgs _
)
```

[C#]

```
public delegate void NntpClient.OnCommandEventHandler(
      object sender,
      CommandEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A CommandEventArgs object that contains the event data.

Remarks

When you create an **OnCommandEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnCommandEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NntpClient (in SocketTools.NntpClient.dll)

See Also

SocketTools Namespace

NntpClient.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub NntpClient.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void NntpClient.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ErrorEventArgs that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NntpClient (in SocketTools.NntpClient.dll)

See Also

SocketTools Namespace

NntpClient.OnProgressEventHandler Delegate

Represents the method that will handle the OnProgress event.

[Visual Basic]

```
Public Delegate Sub NntpClient.OnProgressEventHandler( _
   ByVal sender As Object, _
   ByVal e As ProgressEventArgs _
)
```

[C#]

```
public delegate void NntpClient.OnProgressEventHandler(
      object sender,
      ProgressEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A ProgressEventArgs that contains the event data.

Remarks

When you create an **OnProgressEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnProgressEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NntpClient (in SocketTools.NntpClient.dll)

See Also

SocketTools Namespace

NntpClient.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see NntpClient.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.NntpClient.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class NntpClient.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class NntpClient.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the NntpClient class.

Example

```
<Assembly: SocketTools.NntpClient.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.NntpClient.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.NntpClient (in SocketTools.NntpClient.dll)

See Also

NntpClient.RuntimeLicenseAttribute Members | SocketTools Namespace

NntpClient.RuntimeLicenseAttribute Members

NntpClient.RuntimeLicenseAttribute overview

Public Instance Constructors

NntpClient.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

NntpClient.RuntimeLicenseAttribute Class | SocketTools Namespace

NntpClient.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public NntpClient.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the NntpClient class.

See Also

NntpClient.RuntimeLicenseAttribute Class | SocketTools Namespace

NntpClient.RuntimeLicenseAttribute Properties

The properties of the **NntpClient.RuntimeLicenseAttribute** class are listed below. For a complete list of **NntpClient.RuntimeLicenseAttribute** class members, see the NntpClient.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

NntpClient.RuntimeLicenseAttribute Class | SocketTools Namespace

NntpClient.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

NntpClient.RuntimeLicenseAttribute Class | SocketTools Namespace

NntpClientException Class

The exception that is thrown when a client error occurs.

For a list of all members of this type, see NntpClientException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.NntpClientException

[Visual Basic]
```
Public Class NntpClientException
    Inherits ApplicationException
```

[C#]
```
public class NntpClientException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A NntpClientException is thrown by the NntpClient class when an error occurs.

The default constructor for the NntpClientException class sets the **ErrorCode** property to the last client error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NntpClient (in SocketTools.NntpClient.dll)

See Also

NntpClientException Members | SocketTools Namespace

NntpClientException Members

NntpClientException overview

Public Instance Constructors

NntpClientException	Overloaded. Initializes a new instance of the NntpClientException class.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

NntpClientException Class | SocketTools Namespace

NntpClientException Constructor

Initializes a new instance of the NntpClientException class with the last network error code.

Overload List

Initializes a new instance of the NntpClientException class with the last network error code.

public NntpClientException();

Initializes a new instance of the NntpClientException class with a specified error number.

public NntpClientException(int);

Initializes a new instance of the NntpClientException class with a specified error message.

public NntpClientException(string);

Initializes a new instance of the NntpClientException class with a specified error message and a reference to the inner exception that is the cause of this exception.

public NntpClientException(string,Exception);

See Also

NntpClientException Class | SocketTools Namespace

NntpClientException Constructor ()

Initializes a new instance of the NntpClientException class with the last network error code.

[Visual Basic]

```
Overloads Public Sub New()
```

[C#]

```
public NntpClientException();
```

Remarks

The ctor constructor sets the **ErrorCode** property to the last client error that occurred. For more information about the errors that may occur, refer to the NntpClient.ErrorCode enumeration.

See Also

NntpClientException Class | SocketTools Namespace | NntpClientException Constructor Overload List

NntpClientException Constructor (String)

Initializes a new instance of the NntpClientException class with a specified error message.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]

```
public NntpClientException(
   string message
);
```

Parameters

message

The error message that explains the reason for the exception.

Remarks

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

NntpClientException Class | SocketTools Namespace | NntpClientException Constructor Overload List

NntpClientException Constructor (String, Exception)

Initializes a new instance of the NntpClientException class with a specified error message and a reference to the inner exception that is the cause of this exception.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal innerException As Exception _
)
```

[C#]

```
public NntpClientException(
   string message,
   Exception innerException
);
```

Parameters

message

The error message that explains the reason for the exception.

innerException

The exception that is the cause of the current exception. If the ***innerException*** parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

NntpClientException Class | SocketTools Namespace | NntpClientException Constructor Overload List

NntpClientException Constructor (Int32)

Initializes a new instance of the NntpClientException class with a specified error number.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

[C#]

```
public NntpClientException(
   int code
);
```

Parameters

code
An integer value which specifies an error code.

Remarks

This constructor sets the **ErrorCode** property to the specified error code. For more information about the errors that may occur, refer to the NntpClient.ErrorCode enumeration.

See Also

NntpClientException Class | SocketTools Namespace | NntpClientException Constructor Overload List

NntpClientException Properties

The properties of the **NntpClientException** class are listed below. For a complete list of **NntpClientException** class members, see the NntpClientException Members topic.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

NntpClientException Class | SocketTools Namespace

NntpClientException.ErrorCode Property

Gets a value which specifies the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property ErrorCode As ErrorCode
```

[C#]

```
public NntpClient.ErrorCode ErrorCode {get;}
```

Property Value

Returns a NntpClient.ErrorCode enumeration value which specifies the error code.

Remarks

The **ErrorCode** property returns the error code that specifies the cause of the exception.

The default constructor for the NntpClientException class sets the error code to the last network error that occurred. For more information about the errors that may occur, refer to the NntpClient.ErrorCode enumeration.

See Also

NntpClientException Class | SocketTools Namespace

NntpClientException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]

```
Overrides Public ReadOnly Property Message As String
```

[C#]

```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

NntpClientException Class | SocketTools Namespace

NntpClientException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception. This value is the same as the values returned by the Windows Sockets API. For more information about socket error codes, see the Windows Socket Version 2 API error code documentation in MSDN.

See Also

NntpClientException Class | SocketTools Namespace

NntpClientException Methods

The methods of the **NntpClientException** class are listed below. For a complete list of **NntpClientException** class members, see the NntpClientException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

NntpClientException Class | SocketTools Namespace

NntpClientException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]
```
Overrides Public Function ToString() As String
```

[C#]
```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

NntpClientException Class | SocketTools Namespace

NetworkTime Class

Query a time server for the current time and synchronize the local system clock with that value.

For a list of all members of this type, see NetworkTime Members.

System.Object
  SocketTools.NetworkTime

[Visual Basic]
```
Public Class NetworkTime
    Implements IDisposable
```

[C#]
```
public class NetworkTime : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The NetworkTime class library provides an interface for synchronizing the local system's time and date with that of a remote server. The time values returned are in in Coordinated Universal Time and be adjusted for the local host's timezone. The class enables developers to query a server for the current time and then update the system clock if desired.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NetworkTime (in SocketTools.NetworkTime.dll)

See Also

NetworkTime Members | SocketTools Namespace

NetworkTime Members

NetworkTime overview

Public Static (Shared) Fields

timePortDefault	A constant value which specifies the default port number.
timeTimeout	A constant value which specifies the default timeout period.

Public Instance Constructors

NetworkTime Constructor	Initializes a new instance of the NetworkTime class.

Public Instance Properties

AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets the local Internet address that the client is bound to.
Localize	Gets and sets a value which specifies if the time should be adjusted for the current timezone.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
Options	Gets and sets a value which specifies one or more client options.
RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error

	occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
Version	Gets a value which returns the current version of the NetworkTime class library.

Public Instance Methods

AttachThread	Attach an instance of the class to the current thread
ConvertTime	Overloaded. Convert a network time value to a System.DateTime value.
Dispose	Overloaded. Releases all resources used by NetworkTime.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetTime	Overloaded. Get the current system date and time from the specified time server.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the NetworkTime class.
Reset	Reset the internal state of the object, resetting all properties to their default values.
SetTime	Overloaded. Set the local system clock to the specified System.DateTime value.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.

Public Instance Events

OnCancel	Occurs when a blocking client operation is canceled.
OnError	Occurs when an client operation fails.
OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the NetworkTime class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime Constructor

Initializes a new instance of the NetworkTime class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public NetworkTime();
```

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime Properties

The properties of the **NetworkTime** class are listed below. For a complete list of **NetworkTime** class members, see the NetworkTime Members topic.

Public Instance Properties

AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets the local Internet address that the client is bound to.
Localize	Gets and sets a value which specifies if the time should be adjusted for the current timezone.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
Options	Gets and sets a value which specifies one or more client options.
RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.

TraceFlags	Gets and sets a value which specifies the client function tracing flags.
Version	Gets a value which returns the current version of the NetworkTime class library.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.AutoResolve Property

Gets and sets a value that determines if host names and addresses are automatically resolved.

[Visual Basic]

```
Public Property AutoResolve As Boolean
```

[C#]

```
public bool AutoResolve {get; set;}
```

Property Value

Returns **true** if host names are automatically resolved to Internet addresses. The default value is **false**.

Remarks

Setting the **AutoResolve** property determines if the class automatically resolves host names and addresses specified by the **HostName** and **HostAddress** properties. If set to **true**, setting the **HostName** property will cause the class to automatically determine the corresponding IP address and update the **HostAddress** property accordingly. Likewise, setting the **HostAddress** property will cause the class to determine the host name and update the **HostName** property. Setting this property to **false** prevents the class from resolving host names until a connection attempt is made.

It is important to note that setting the **HostName** or **HostAddress** property may cause the current thread to block, sometimes for several seconds, until the name or address is resolved. To prevent this behavior, set this property value to **false**.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.HostAddress Property

Gets and sets a value which specifies the Internet address used to establish a connection.

[Visual Basic]
```
Public Property HostAddress As String
```

[C#]
```
public string HostAddress {get; set;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **HostAddress** property can be used to set the Internet address for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the address is assigned to a valid host name, the **HostName** property will be updated with that value.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.HostName Property

Gets and sets a value which specifies the host name used to establish a connection.

[Visual Basic]

```
Public Property HostName As String
```

[C#]

```
public string HostName {get; set;}
```

Property Value

A string which specifies a host name.

Remarks

The **HostName** property can be used to set the host name for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the name can be resolved to a valid Internet address, the **HostAddress** property will be updated with that value.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.IsInitialized Property

Gets a value which indicates if the current instance of the class has been initialized successfully.

[Visual Basic]
```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]
```
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the class instance has been initialized; otherwise returns **false**.

Remarks

The **IsInitialized** property is used to determine if the current instance of the class has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the class may not be able to initialize itself.

The most common reasons that a class instance may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required networking libraries or not being able to access the system registry.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public NetworkTime.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.LocalAddress Property

Gets the local Internet address that the client is bound to.

[Visual Basic]

```
Public ReadOnly Property LocalAddress As String
```

[C#]

```
public string LocalAddress {get;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **LocalAddress** property returns the local Internet address that the client is bound to when a connection is established with a remote host. This property may return either an IPv4 or IPv6 formatted address, depending on the type of connection that was established.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.Localize Property

Gets and sets a value which specifies if the time should be adjusted for the current timezone.

[Visual Basic]

```
Public Property Localize As Boolean
```

[C#]

```
public bool Localize {get; set;}
```

Property Value

A boolean value which specifies if the time should be adjusted for the local timezone.

Remarks

The **Localize** property controls how date and time values are localized when the **GetTime** method is called. If the property is set to **true** the date and time will be adjusted to the current timezone. If the property is set to **false** the file date and time are returned as UTC (Coordinated Universal Time) values.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.LocalName Property

Gets a value which specifies the host name for the local system.

[Visual Basic]

```
Public ReadOnly Property LocalName As String
```

[C#]

```
public string LocalName {get;}
```

Property Value

A string which specifies the local host name.

Remarks

The **LocalName** property returns the fully-qualified host name assigned to the local system. If the system has not been configured with an Internet domain name, then this property will return the NetBIOS name assigned to the local system.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.LocalPort Property

Gets the local port number the client is bound to.

[Visual Basic]

```
Public ReadOnly Property LocalPort As Integer
```

[C#]

```
public int LocalPort {get;}
```

Property Value

An integer value which specifies a port number. The default value is 0.

Remarks

The **LocalPort** property is used to identify the local port number that the client is bound to to when a connection is established with a remote host.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.RemotePort Property

Gets and sets a value which specifies the remote port number.

[Visual Basic]

```
Public Property RemotePort As Integer
```

[C#]

```
public int RemotePort {get; set;}
```

Property Value

An integer value which specifies a port number.

Remarks

The **RemotePort** property is used to set the port number that will be used to establish a connection with a remote host. If the port number specifies a well-known port, the **RemoteService** property will be updated with that name.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.ThreadModel Property

Gets and sets a value which specifies the threading model for the class instance.

[Visual Basic]
```
Public Property ThreadModel As ThreadingModel
```

[C#]
```
public NetworkTime.ThreadingModel ThreadModel {get; set;}
```

Property Value

Returns one or more ThreadingModel enumeration value which specifies the threading model for the client. The default value for this property is **modelSingleThread**.

Remarks

The **ThreadModel** property specifies the threading model that is used by the class instance when a connection is established. The default value for this property is **modelSingleThread**, which specifies that only the thread that established the connection should be permitted to invoke methods. It is important to note that this threading model does not limit the application to a single thread of execution. When a session is established using the **Connect** method, that session is attached to the thread that created it. From that point on, until the session is terminated, only the owner may invoke methods in that instance of the class. The ownership of the class instance may be transferred from one thread to another using the **AttachThread** method.

Setting this property to **modelFreeThread** disables certain internal safety checks that are performed by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads. The application must ensure that no two threads will attempt to invoke a blocking method at the same time. In other words, if one thread invokes a method, the application must ensure that another thread will not attempt to invoke any other method at the same time using the same instance of the class.

Changing the value of this property will not affect an active client session. The threading model must be specified prior to invoking the **Connect** method.

See Also

NetworkTime Class | SocketTools Namespace | AttachThread Method | ThreadingModel Enumeration | ThreadModel Attribute

NetworkTime.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]

```
Public Property ThrowError As Boolean
```

[C#]

```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.Timeout Property

Gets and sets a value which specifies a timeout period in seconds.

[Visual Basic]

```
Public Property Timeout As Integer
```

[C#]

```
public int Timeout {get; set;}
```

Property Value

An integer value which specifies a timeout period in seconds.

Remarks

Setting the **Timeout** property specifies the number of seconds until a blocking operation fails and returns an error.

The timeout period is only used when the client is in blocking mode. Although this property can be changed when the client is in non-blocking mode, the value will be ignored until the client is returned to blocking mode.

For most applications it is recommended the timeout period be set between 10 and 20 seconds.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.Trace Property

Gets and sets a value which indicates if network function logging is enabled.

[Visual Basic]
```
Public Property Trace As Boolean
```

[C#]
```
public bool Trace {get; set;}
```

Property Value

Returns **true** if network function tracing is enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Trace** property is used to enable (or disable) the tracing of network function calls. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the SocketTools networking classes and is enabled or disabled for an entire process. This means that once trace logging is enabled for a given component, all of the function calls made by the process using any of the SocketTools classes will be logged. For example, if you have an application using both the File Transfer Protocol and Post Office Protocol classes, and you set the **Trace** property to **true**, function calls made by both classes will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all classes used by the process.

If trace logging is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the logfile is fairly large. Since logfiles can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

When redistributing your application, make sure that you include the **SocketTools10.TraceLog.dll** module with your installation. If this library is not present, then no trace output will be generated and the value of the **Trace** property will be ignored. Only those function calls made by the SocketTools networking classes will be logged. Calls made directly to the Windows Sockets API, or calls made by other classes, will not be logged.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.TraceFile Property

Gets and sets a value which specifies the name of the logfile.

[Visual Basic]

```
Public Property TraceFile As String
```

[C#]

```
public string TraceFile {get; set;}
```

Property Value

A string which specifies the name of the file.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named **SocketTools.log** is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFile** property should be set to the same value for each instance. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
 MyApp INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0 MyApp WRN:
connect(46, 192.0.0.1:1234, 16) returned -1 [10035] MyApp ERR: accept(46,
NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the control is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Note that if the specified file cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.TraceFlags Property

Gets and sets a value which specifies the client function tracing flags.

[Visual Basic]

```
Public Property TraceFlags As TraceOptions
```

[C#]

```
public NetworkTime.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration which specifies the amount of detail written to the trace logfile.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Because network function tracing is enabled per-process, the trace flags are shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFlags** property should be set to the same value for each instance. Changing the trace flags for any one instance of the class will affect the logging performed for all SocketTools classes used by the application.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being written and the error **errorOperationWouldBlock** occurs, a warning is generated because the application simply needs to attempt to write the data at a later time.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.Version Property

Gets a value which returns the current version of the NetworkTime class library.

[Visual Basic]
```
Public ReadOnly Property Version As String
```

[C#]
```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the NetworkTime class library. This value can be used by an application for validation and debugging purposes.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime Methods

The methods of the **NetworkTime** class are listed below. For a complete list of **NetworkTime** class members, see the NetworkTime Members topic.

Public Instance Methods

AttachThread	Attach an instance of the class to the current thread
ConvertTime	Overloaded. Convert a network time value to a System.DateTime value.
Dispose	Overloaded. Releases all resources used by NetworkTime.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetTime	Overloaded. Get the current system date and time from the specified time server.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the NetworkTime class.
Reset	Reset the internal state of the object, resetting all properties to their default values.
SetTime	Overloaded. Set the local system clock to the specified System.DateTime value.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the NetworkTime class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.ConvertTime Method

Convert a System.DateTime value to a network time value.

Overload List

Convert a System.DateTime value to a network time value.

public bool ConvertTime(ref DateTime,ref long);

Convert a network time value to a System.DateTime value.

public bool ConvertTime(long,ref DateTime);

Convert a network time value to a System.DateTime value.

public bool ConvertTime(long,ref DateTime,bool);

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.ConvertTime Method (DateTime, Int64)

Convert a System.DateTime value to a network time value.

[Visual Basic]

```
Overloads Public Function ConvertTime( _
   ByRef dateTime As Date, _
   ByRef networkTime As Long _
) As Boolean
```

[C#]

```
public bool ConvertTime(
   ref DateTime dateTime,
   ref long networkTime
);
```

Parameters

dateTime

A **System.DateTime** structure passed by reference which will contain the date and time when the method returns.

networkTime

A long integer value which specifies the network time.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

Network time is expressed as the number of seconds that has elapsed since midnight, January 1, 1900 UTC.

See Also

NetworkTime Class | SocketTools Namespace | NetworkTime.ConvertTime Overload List

NetworkTime.ConvertTime Method (Int64, DateTime)

Convert a network time value to a System.DateTime value.

[Visual Basic]

```
Overloads Public Function ConvertTime( _
   ByVal networkTime As Long, _
   ByRef dateTime As Date _
) As Boolean
```

[C#]

```
public bool ConvertTime(
   long networkTime,
   ref DateTime dateTime
);
```

Parameters

networkTime

A long integer value which specifies the network time.

dateTime

A **System.DateTime** structure passed by reference which will contain the date and time when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

If the **Localize** property is set to **true**, the network time value will be localized to the current timezone. If the property value is **false** the network time will be converted as-is.

Network time is expressed as the number of seconds that has elapsed since midnight, January 1, 1900 UTC.

See Also

NetworkTime Class | SocketTools Namespace | NetworkTime.ConvertTime Overload List

NetworkTime.ConvertTime Method (Int64, DateTime, Boolean)

Convert a network time value to a System.DateTime value.

[Visual Basic]

```
Overloads Public Function ConvertTime( _
   ByVal networkTime As Long, _
   ByRef dateTime As Date, _
   ByVal bLocalize As Boolean _
) As Boolean
```

[C#]

```
public bool ConvertTime(
   long networkTime,
   ref DateTime dateTime,
   bool bLocalize
);
```

Parameters

networkTime
A long integer value which specifies the network time.

dateTime
A **System.DateTime** structure passed by reference which will contain the date and time when the method returns.

bLocalize
A boolean value which specifies if the network time value should be localized for the current timezone.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

Network time is expressed as the number of seconds that has elapsed since midnight, January 1, 1900 UTC.

See Also

NetworkTime Class | SocketTools Namespace | NetworkTime.ConvertTime Overload List

NetworkTime.Dispose Method

Releases all resources used by NetworkTime.

Overload List

Releases all resources used by NetworkTime.

public void Dispose();

Releases the unmanaged resources allocated by the NetworkTime class and optionally releases the managed resources.

protected virtual void Dispose(bool);

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.Dispose Method ()

Releases all resources used by NetworkTime.

[Visual Basic]

```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]

```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

NetworkTime Class | SocketTools Namespace | NetworkTime.Dispose Overload List

NetworkTime.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the NetworkTime class and optionally releases the managed resources.

[Visual Basic]

```
Overridable Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]

```
protected virtual void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **NetworkTime** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

NetworkTime Class | SocketTools Namespace | NetworkTime.Dispose Overload List

NetworkTime.Finalize Method

Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.

[Visual Basic]
```
Overrides Protected Sub Finalize()
```

[C#]
```
protected override void Finalize();
```

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.GetTime Method

Get the current system date and time from a default time server.

Overload List

Get the current system date and time from a default time server.

public bool GetTime(ref DateTime);

Get the current system date and time from the time server.

public bool GetTime(ref long);

Get the current system date and time from the specified time server.

public bool GetTime(string,ref DateTime);

Get the current system date and time from the specified time server.

public bool GetTime(string,int,ref DateTime);

Get the current system date and time from the specified time server.

public bool GetTime(string,int,int,ref DateTime);

Get the current system date and time from the specified time server.

public bool GetTime(string,int,int,ref long);

Get the current system date and time from the specified time server.

public bool GetTime(string,int,ref long);

Get the current system date and time from the specified time server.

public bool GetTime(string,ref long);

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.GetTime Method (DateTime)

Get the current system date and time from a default time server.

[Visual Basic]

```
Overloads Public Function GetTime( _
   ByRef dateTime As Date _
) As Boolean
```

[C#]

```
public bool GetTime(
   ref DateTime dateTime
);
```

Parameters

dateTime

A **System.DateTime** structure passed by reference which will contain the date and time returned by the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The time and date retrieved from the server will be returned as a **System.DateTime** value according to the user's current locale.

If the **HostName** or **HostAddress** property does not specify a server host name or address, a public timeserver will be automatically selected by default.

See Also

NetworkTime Class | SocketTools Namespace | NetworkTime.GetTime Overload List

NetworkTime.GetTime Method (Int64)

Get the current system date and time from the time server.

[Visual Basic]

```
Overloads Public Function GetTime( _
   ByRef networkTime As Long _
) As Boolean
```

[C#]

```
public bool GetTime(
   ref long networkTime
);
```

Parameters

networkTime

A **System.DateTime** structure passed by reference which will contain the date and time returned by the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

Network time is expressed as the number of seconds that has elapsed since midnight, January 1, 1900 UTC.

If the **HostName** or **HostAddress** property does not specify a server host name or address, a public timeserver will be automatically selected by default.

See Also

NetworkTime Class | SocketTools Namespace | NetworkTime.GetTime Overload List

NetworkTime.GetTime Method (String, DateTime)

Get the current system date and time from the specified time server.

```
[Visual Basic]
Overloads Public Function GetTime( _
   ByVal hostName As String, _
   ByRef dateTime As Date _
) As Boolean
```

```
[C#]
public bool GetTime(
   string hostName,
   ref DateTime dateTime
);
```

Parameters

hostName
A string which specifies the host name or address of the time server.

dateTime
A **System.DateTime** structure passed by reference which will contain the date and time returned by the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The time and date retrieved from the server will be returned as a **System.DateTime** value according to the user's current locale.

In the United States, the National Institute of Standards and Technology (NIST) hosts a number of public servers which can be used to obtain the current time:

Server Name	Location
time-a.nist.gov	Gaithersburg, Maryland
time-b.nist.gov	Gaithersburg, Maryland
time-nw.nist.gov	Redmond, Washington
time-a.timefreq.bldrdoc.gov	Boulder, Colorado
time-b.timefreq.bldrdoc.gov	Boulder, Colorado
time-c.timefreq.bldrdoc.gov	Boulder, Colorado

Time servers are also commonly maintained by Internet service providers and universities. If you are unable to obtain the time from a server, contact the system administrator to determine if they have the standard time service available on port 37.

See Also

NetworkTime Class | SocketTools Namespace | NetworkTime.GetTime Overload List

NetworkTime.GetTime Method (String, Int32, DateTime)

Get the current system date and time from the specified time server.

[Visual Basic]

```
Overloads Public Function GetTime( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByRef dateTime As Date _
) As Boolean
```

[C#]

```
public bool GetTime(
   string hostName,
   int hostPort,
   ref DateTime dateTime
);
```

Parameters

hostName
A string which specifies the host name or address of the time server.

hostPort
An integer value which specifies the port to connect to on the server. The default port value for a time server is port 37.

dateTime
A **System.DateTime** structure passed by reference which will contain the date and time returned by the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The time and date retrieved from the server will be returned as a **System.DateTime** value according to the user's current locale.

In the United States, the National Institute of Standards and Technology (NIST) hosts a number of public servers which can be used to obtain the current time:

Server Name	Location
time-a.nist.gov	Gaithersburg, Maryland
time-b.nist.gov	Gaithersburg, Maryland
time-nw.nist.gov	Redmond, Washington
time-a.timefreq.bldrdoc.gov	Boulder, Colorado
time-b.timefreq.bldrdoc.gov	Boulder, Colorado
time-c.timefreq.bldrdoc.gov	Boulder, Colorado

Time servers are also commonly maintained by Internet service providers and universities. If you are unable to obtain the time from a server, contact the system administrator to determine if they have the standard time service available on port 37.

See Also

NetworkTime Class | SocketTools Namespace | NetworkTime.GetTime Overload List

NetworkTime.GetTime Method (String, Int32, Int32, DateTime)

Get the current system date and time from the specified time server.

[Visual Basic]

```
Overloads Public Function GetTime( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer, _
   ByRef dateTime As Date _
) As Boolean
```

[C#]

```
public bool GetTime(
   string hostName,
   int hostPort,
   int timeout,
   ref DateTime dateTime
);
```

Parameters

hostName

A string which specifies the host name or address of the time server.

hostPort

An integer value which specifies the port to connect to on the server. The default port value for a time server is port 37.

timeout

An integer value which specifies the number of seconds until a blocking operation fails and returns an error.

dateTime

A **System.DateTime** structure passed by reference which will contain the date and time returned by the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The time and date retrieved from the server will be returned as a **System.DateTime** value according to the user's current locale.

In the United States, the National Institute of Standards and Technology (NIST) hosts a number of public servers which can be used to obtain the current time:

Server Name	Location
time-a.nist.gov	Gaithersburg, Maryland
time-b.nist.gov	Gaithersburg, Maryland
time-nw.nist.gov	Redmond, Washington
time-a.timefreq.bldrdoc.gov	Boulder, Colorado
time-b.timefreq.bldrdoc.gov	Boulder, Colorado

time-c.timefreq.bldrdoc.gov	Boulder, Colorado

Time servers are also commonly maintained by Internet service providers and universities. If you are unable to obtain the time from a server, contact the system administrator to determine if they have the standard time service available on port 37.

See Also

NetworkTime Class | SocketTools Namespace | NetworkTime.GetTime Overload List

NetworkTime.GetTime Method (String, Int32, Int32, Int64)

Get the current system date and time from the specified time server.

[Visual Basic]

```
Overloads Public Function GetTime( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer, _
   ByRef networkTime As Long _
) As Boolean
```

[C#]

```
public bool GetTime(
   string hostName,
   int hostPort,
   int timeout,
   ref long networkTime
);
```

Parameters

hostName

A string which specifies the host name or address of the time server.

hostPort

An integer value which specifies the port to connect to on the server. The default port value for a time server is port 37.

timeout

An integer value which specifies the number of seconds until a blocking operation fails and returns an error. ///

networkTime

A long integer passed by reference which will contain the network time value returned by the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

Network time is expressed as the number of seconds that has elapsed since midnight, January 1, 1900 UTC.

In the United States, the National Institute of Standards and Technology (NIST) hosts a number of public servers which can be used to obtain the current time:

Server Name	Location
time-a.nist.gov	Gaithersburg, Maryland
time-b.nist.gov	Gaithersburg, Maryland
time-nw.nist.gov	Redmond, Washington
time-a.timefreq.bldrdoc.gov	Boulder, Colorado
time-b.timefreq.bldrdoc.gov	Boulder, Colorado
time-c.timefreq.bldrdoc.gov	Boulder, Colorado

Time servers are also commonly maintained by Internet service providers and universities. If you are unable to obtain the time from a server, contact the system administrator to determine if they have the standard time service available on port 37.

See Also

NetworkTime Class | SocketTools Namespace | NetworkTime.GetTime Overload List

NetworkTime.GetTime Method (String, Int32, Int64)

Get the current system date and time from the specified time server.

[Visual Basic]

```
Overloads Public Function GetTime( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByRef networkTime As Long _
) As Boolean
```

[C#]

```
public bool GetTime(
   string hostName,
   int hostPort,
   ref long networkTime
);
```

Parameters

hostName
A string which specifies the host name or address of the time server.

hostPort
An integer value which specifies the port to connect to on the server. The default port value for a time server is port 37.

networkTime
A long integer passed by reference which will contain the network time value returned by the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

Network time is expressed as the number of seconds that has elapsed since midnight, January 1, 1900 UTC.

In the United States, the National Institute of Standards and Technology (NIST) hosts a number of public servers which can be used to obtain the current time:

Server Name	Location
time-a.nist.gov	Gaithersburg, Maryland
time-b.nist.gov	Gaithersburg, Maryland
time-nw.nist.gov	Redmond, Washington
time-a.timefreq.bldrdoc.gov	Boulder, Colorado
time-b.timefreq.bldrdoc.gov	Boulder, Colorado
time-c.timefreq.bldrdoc.gov	Boulder, Colorado

Time servers are also commonly maintained by Internet service providers and universities. If you are unable to obtain the time from a server, contact the system administrator to determine if they have the standard time service available on port 37.

See Also

NetworkTime Class | SocketTools Namespace | NetworkTime.GetTime Overload List

NetworkTime.GetTime Method (String, Int64)

Get the current system date and time from the specified time server.

```
[Visual Basic]
Overloads Public Function GetTime( _
   ByVal hostName As String, _
   ByRef networkTime As Long _
) As Boolean
```

```
[C#]
public bool GetTime(
   string hostName,
   ref long networkTime
);
```

Parameters

hostName
A string which specifies the host name or address of the time server.

networkTime
A long integer passed by reference which will contain the network time value returned by the server.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

Network time is expressed as the number of seconds that has elapsed since midnight, January 1, 1900 UTC.

In the United States, the National Institute of Standards and Technology (NIST) hosts a number of public servers which can be used to obtain the current time:

Server Name	Location
time-a.nist.gov	Gaithersburg, Maryland
time-b.nist.gov	Gaithersburg, Maryland
time-nw.nist.gov	Redmond, Washington
time-a.timefreq.bldrdoc.gov	Boulder, Colorado
time-b.timefreq.bldrdoc.gov	Boulder, Colorado
time-c.timefreq.bldrdoc.gov	Boulder, Colorado

Time servers are also commonly maintained by Internet service providers and universities. If you are unable to obtain the time from a server, contact the system administrator to determine if they have the standard time service available on port 37.

See Also

NetworkTime Class | SocketTools Namespace | NetworkTime.GetTime Overload List

NetworkTime.Initialize Method

Initialize an instance of the NetworkTime class.

Overload List

Initialize an instance of the NetworkTime class.

public bool Initialize();

Initialize an instance of the NetworkTime class.

public bool Initialize(string);

See Also

NetworkTime Class | SocketTools Namespace | Uninitialize Method

NetworkTime.Initialize Method ()

Initialize an instance of the NetworkTime class.

[Visual Basic]
```
Overloads Public Function Initialize() As Boolean
```

[C#]
```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the NetworkTime class, loading the networking library and allocating resources for the current thread. Typically it is not necessary to explicitly call this method because the instance of the class is initialized by the class constructor. However, if the **Uninitialize** method is called, the class must be re-initialized before any other methods are called.

See Also

NetworkTime Class | SocketTools Namespace | NetworkTime.Initialize Overload List | Uninitialize Method

NetworkTime.Initialize Method (String)

Initialize an instance of the NetworkTime class.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the NetworkTime class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the NetworkTime class.

Example

The following example shows how to use the Initialize method to initialize an instance of the class. This example assumes that the license key string has been defined in code.

```
SocketTools.NetworkTime timeClient = new SocketTools.NetworkTime();

if (timeClient.Initialize(strLicenseKey) == false)
{
    MessageBox.Show(timeClient.LastErrorString, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    return;
}
```

```
Dim timeClient As New SocketTools.NetworkTime

If timeClient.Initialize(strLicenseKey) = False Then
    MsgBox(timeClient.LastErrorString, vbIconExclamation)
    Exit Sub
End If
```

See Also

NetworkTime Class | SocketTools Namespace | NetworkTime.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

NetworkTime.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. If a socket has been allocated, it will be released and any active connections will be terminated. All properties will be reset to their default values.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.SetTime Method

Set the local system clock to the current date and time as returned by the default timeserver.

Overload List

Set the local system clock to the current date and time as returned by the default timeserver.

public bool SetTime();

Set the local system clock to the specified System.DateTime value.

public bool SetTime(DateTime);

Set the local system clock to the specified network time value.

public bool SetTime(long);

Set the local system clock to the current date and time as returned by the specified timeserver.

public bool SetTime(string);

Set the local system clock to the current date and time as returned by the specified timeserver.

public bool SetTime(string,int);

Set the local system clock to the current date and time as returned by the specified timeserver.

public bool SetTime(string,int,int);

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.SetTime Method (DateTime)

Set the local system clock to the specified System.DateTime value.

[Visual Basic]

```
Overloads Public Function SetTime( _
   ByVal dateTime As Date _
) As Boolean
```

[C#]

```
public bool SetTime(
   DateTime dateTime
);
```

Parameters

dateTime

A **System.DateTime** structure which specifies the date and time the local system clock should be set to.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SetTime** method causes the class to update the local system's clock to the specified date and time. It is required that the user have the appropriate privileges required to change the system clock, otherwise an error will be returned.

See Also

NetworkTime Class | SocketTools Namespace | NetworkTime.SetTime Overload List

NetworkTime.SetTime Method (Int64)

Set the local system clock to the specified network time value.

[Visual Basic]

```
Overloads Public Function SetTime( _
   ByVal networkTime As Long _
) As Boolean
```

[C#]

```
public bool SetTime(
   long networkTime
);
```

Parameters

networkTime

A long integer which specifies the date and time the local system clock should be set to.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SetTime** method causes the class to update the local system's clock to the specified date and time. It is required that the user have the appropriate privileges required to change the system clock, otherwise an error will be returned.

Network time is expressed as the number of seconds that has elapsed since midnight, January 1, 1900 UTC.

See Also

NetworkTime Class | SocketTools Namespace | NetworkTime.SetTime Overload List

NetworkTime.Uninitialize Method

Uninitialize the class library and release any resources allocated for the current thread.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method terminates any active connection, releases resources allocated for the current thread and unloads the networking library. After this method has been called, no further client operations may be performed until the class instance has been re-initialized.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the class is no longer needed.

See Also

NetworkTime Class | SocketTools Namespace | Initialize Method

NetworkTime Events

The events of the **NetworkTime** class are listed below. For a complete list of **NetworkTime** class members, see the NetworkTime Members topic.

Public Instance Events

⚡ OnCancel	Occurs when a blocking client operation is canceled.
⚡ OnError	Occurs when an client operation fails.
⚡ OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.OnCancel Event

Occurs when a blocking client operation is canceled.

[Visual Basic]

```
Public Event OnCancel As EventHandler
```

[C#]

```
public event EventHandler OnCancel;
```

Remarks

The **OnCancel** event is generated when a blocking client operation, such as sending or receiving data, is canceled with the **Cancel** method. To assist in determining which operation was canceled, check the value of the **Status** property.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.OnError Event

Occurs when an client operation fails.

```
[Visual Basic]
Public Event OnError As OnErrorEventHandler
```

```
[C#]
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type NetworkTime.ErrorEventArgs containing data related to this event. The following **NetworkTime.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a client operation fails.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see NetworkTime.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.NetworkTime.ErrorEventArgs

[Visual Basic]

```
Public Class NetworkTime.ErrorEventArgs
    Inherits EventArgs
```

[C#]

```
public class NetworkTime.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NetworkTime (in SocketTools.NetworkTime.dll)

See Also

NetworkTime.ErrorEventArgs Members | SocketTools Namespace

NetworkTime.ErrorEventArgs Members

NetworkTime.ErrorEventArgs overview

Public Instance Constructors

NetworkTime.ErrorEventArgs Constructor	Initializes a new instance of the NetworkTime.ErrorEventArgs class.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

NetworkTime.ErrorEventArgs Class | SocketTools Namespace

NetworkTime.ErrorEventArgs Constructor

Initializes a new instance of the NetworkTime.ErrorEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public NetworkTime.ErrorEventArgs();
```

See Also

NetworkTime.ErrorEventArgs Class | SocketTools Namespace

NetworkTime.ErrorEventArgs Properties

The properties of the **NetworkTime.ErrorEventArgs** class are listed below. For a complete list of **NetworkTime.ErrorEventArgs** class members, see the NetworkTime.ErrorEventArgs Members topic.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

NetworkTime.ErrorEventArgs Class | SocketTools Namespace

NetworkTime.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

NetworkTime.ErrorEventArgs Class | SocketTools Namespace | Error Property

NetworkTime.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public NetworkTime.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

NetworkTime.ErrorEventArgs Class | SocketTools Namespace | Description Property

NetworkTime.OnTimeout Event

Occurs when a blocking operation fails to complete before the timeout period elapses.

[Visual Basic]

```
Public Event OnTimeout As EventHandler
```

[C#]

```
public event EventHandler OnTimeout;
```

Remarks

The **OnTimeout** event occurs when a blocking operation, such as sending or receiving data on the client, fails to complete before the specified timeout period elapses. The timeout period for a blocking operation can be adjusted by setting the **Timeout** property.

This event is only generated if the client is in blocking mode.

See Also

NetworkTime Class | SocketTools Namespace

NetworkTime.ErrorCode Enumeration

Specifies the error codes returned by the NetworkTime class.

[Visual Basic]
```
Public Enum NetworkTime.ErrorCode
```

[C#]
```
public enum NetworkTime.ErrorCode
```

Remarks

The NetworkTime class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.
errorProxyAuthenticationRequired	Proxy authentication required.

errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.
errorUnsupportedMediaType	Request specified an unsupported media type.

errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.
errorNoMoreMailboxes	No more mailboxes exist on this server.

errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.
errorInvalidThreadHandle	The specified thread handle is no longer valid.

errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been defined.

errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or unrecognized.

errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NetworkTime (in SocketTools.NetworkTime.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

NetworkTime.TraceOptions Enumeration

Specifies the logging options that the NetworkTime class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum NetworkTime.TraceOptions
```

```
[C#]
[Flags]
public enum NetworkTime.TraceOptions
```

Remarks

The NetworkTime class uses the **TraceOptions** enumeration to specify what kind of debugging information is written to the trace logfile. These options are only meaningful when trace logging is enabled by setting the **Trace** property to **true**.

Members

Member Name	Description	Value
traceDefault	The default trace logging option. This is the same as specifying the **traceInfo** option.	0
traceInfo	All network function calls are written to the trace file. This is the default value.	0
traceError	Only those network function calls which fail are recorded in the trace file.	1
traceWarning	Only those network function calls which fail, or return values which indicate a warning, are recorded in the trace file.	2
traceHexDump	All network function calls are written to the trace file, plus all the data that is sent or received is displayed, in both ASCII and hexadecimal format.	4
traceProcess	All function calls in the current process are logged, rather than only those functions in the current thread. This option is useful for multithreaded applications that are using worker threads.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.NetworkTime (in SocketTools.NetworkTime.dll)

See Also

SocketTools Namespace

NetworkTime.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub NetworkTime.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void NetworkTime.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender
The source of the event.

e
An ErrorEventArgs that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NetworkTime (in SocketTools.NetworkTime.dll)

See Also

SocketTools Namespace

NetworkTime.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see NetworkTime.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.NetworkTime.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class NetworkTime.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class NetworkTime.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the NetworkTime class.

Example

```
<Assembly: SocketTools.NetworkTime.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.NetworkTime.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.NetworkTime (in SocketTools.NetworkTime.dll)

See Also

NetworkTime.RuntimeLicenseAttribute Members | SocketTools Namespace

NetworkTime.RuntimeLicenseAttribute Members

NetworkTime.RuntimeLicenseAttribute overview

Public Instance Constructors

NetworkTime.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

NetworkTime.RuntimeLicenseAttribute Class | SocketTools Namespace

NetworkTime.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public NetworkTime.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the NetworkTime class.

See Also

NetworkTime.RuntimeLicenseAttribute Class | SocketTools Namespace

NetworkTime.RuntimeLicenseAttribute Properties

The properties of the **NetworkTime.RuntimeLicenseAttribute** class are listed below. For a complete list of **NetworkTime.RuntimeLicenseAttribute** class members, see the NetworkTime.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

NetworkTime.RuntimeLicenseAttribute Class | SocketTools Namespace

NetworkTime.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

NetworkTime.RuntimeLicenseAttribute Class | SocketTools Namespace

NetworkTimeException Class

The exception that is thrown when a client error occurs.

For a list of all members of this type, see NetworkTimeException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.NetworkTimeException

[Visual Basic]

```
Public Class NetworkTimeException
    Inherits ApplicationException
```

[C#]

```
public class NetworkTimeException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A NetworkTimeException is thrown by the NetworkTime class when an error occurs.

The default constructor for the NetworkTimeException class sets the **ErrorCode** property to the last client error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NetworkTime (in SocketTools.NetworkTime.dll)

See Also

NetworkTimeException Members | SocketTools Namespace

NetworkTimeException Members

NetworkTimeException overview

Public Instance Constructors

NetworkTimeException	Overloaded. Initializes a new instance of the NetworkTimeException class.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

NetworkTimeException Class | SocketTools Namespace

NetworkTimeException Constructor

Initializes a new instance of the NetworkTimeException class with the last network error code.

Overload List

Initializes a new instance of the NetworkTimeException class with the last network error code.

public NetworkTimeException();

Initializes a new instance of the NetworkTimeException class with a specified error number.

public NetworkTimeException(int);

Initializes a new instance of the NetworkTimeException class with a specified error message.

public NetworkTimeException(string);

Initializes a new instance of the NetworkTimeException class with a specified error message and a reference to the inner exception that is the cause of this exception.

public NetworkTimeException(string,Exception);

See Also

NetworkTimeException Class | SocketTools Namespace

NetworkTimeException Constructor ()

Initializes a new instance of the NetworkTimeException class with the last network error code.

[Visual Basic]

```
Overloads Public Sub New()
```

[C#]

```
public NetworkTimeException();
```

Remarks

The ctor constructor sets the **ErrorCode** property to the last client error that occurred. For more information about the errors that may occur, refer to the NetworkTime.ErrorCode enumeration.

See Also

NetworkTimeException Class | SocketTools Namespace | NetworkTimeException Constructor Overload List

NetworkTimeException Constructor (String)

Initializes a new instance of the NetworkTimeException class with a specified error message.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]

```
public NetworkTimeException(
   string message
);
```

Parameters

message

The error message that explains the reason for the exception.

Remarks

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

NetworkTimeException Class | SocketTools Namespace | NetworkTimeException Constructor Overload List

NetworkTimeException Constructor (String, Exception)

Initializes a new instance of the NetworkTimeException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal innerException As Exception _
)
```

```
[C#]
public NetworkTimeException(
   string message,
   Exception innerException
);
```

Parameters

message

The error message that explains the reason for the exception.

innerException

The exception that is the cause of the current exception. If the ***innerException*** parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

NetworkTimeException Class | SocketTools Namespace | NetworkTimeException Constructor Overload List

NetworkTimeException Constructor (Int32)

Initializes a new instance of the NetworkTimeException class with a specified error number.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

[C#]

```
public NetworkTimeException(
   int code
);
```

Parameters

code

An integer value which specifies an error code.

Remarks

This constructor sets the **ErrorCode** property to the specified error code. For more information about the errors that may occur, refer to the NetworkTime.ErrorCode enumeration.

See Also

NetworkTimeException Class | SocketTools Namespace | NetworkTimeException Constructor Overload List

NetworkTimeException Properties

The properties of the **NetworkTimeException** class are listed below. For a complete list of **NetworkTimeException** class members, see the NetworkTimeException Members topic.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

NetworkTimeException Class | SocketTools Namespace

NetworkTimeException.ErrorCode Property

Gets a value which specifies the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property ErrorCode As ErrorCode
```

[C#]

```
public NetworkTime.ErrorCode ErrorCode {get;}
```

Property Value

Returns a NetworkTime.ErrorCode enumeration value which specifies the error code.

Remarks

The **ErrorCode** property returns the error code that specifies the cause of the exception.

The default constructor for the NetworkTimeException class sets the error code to the last network error that occurred. For more information about the errors that may occur, refer to the NetworkTime.ErrorCode enumeration.

See Also

NetworkTimeException Class | SocketTools Namespace

NetworkTimeException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]

```
Overrides Public ReadOnly Property Message As String
```

[C#]

```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

NetworkTimeException Class | SocketTools Namespace

NetworkTimeException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception. This value is the same as the values returned by the Windows Sockets API. For more information about socket error codes, see the Windows Socket Version 2 API error code documentation in MSDN.

See Also

NetworkTimeException Class | SocketTools Namespace

NetworkTimeException Methods

The methods of the **NetworkTimeException** class are listed below. For a complete list of **NetworkTimeException** class members, see the NetworkTimeException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

NetworkTimeException Class | SocketTools Namespace

NetworkTimeException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]

```
Overrides Public Function ToString() As String
```

[C#]

```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

NetworkTimeException Class | SocketTools Namespace

NewsFeed Class

Retrieve and process the contents of a syndicated news feed.

For a list of all members of this type, see NewsFeed Members.

System.Object
  SocketTools.NewsFeed

[Visual Basic]
```
Public Class NewsFeed
    Implements IDisposable
```

[C#]
```
public class NewsFeed : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

Really Simple Syndication (RSS) is a collection of standardized formats that are used to publish information about content that is frequently changed. A news feed is published in XML format, which contains one or more items that includes summary text, hyperlinks to source content and additional metadata that is used to describe the item. News feeds can be used for a variety of purposes, including providing updates for weblogs, news headlines, video and audio content. RSS can also be used for other purposes, such as a software updates, where new updates are listed as items in the feed.

News feeds can be accessed remotely from a web server, or locally as an XML formatted text file. The source of the feed is determined by the URI scheme that is specified. If the http or https scheme is specified, then the feed is retrieved from a web server. If the file scheme is used, the feed is considered to be local and is accessed from the disk or local network. The **SocketTools.NewsFeed** class provides an interface that enables you to open a feed by URL and iterate through each of the items in the feed or search for a specific feed item. The class can also be used to parse a string that contains XML data in RSS format, where the feed may have been retrieved from other sources such as a database.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NewsFeed (in SocketTools.NewsFeed.dll)

See Also

NewsFeed Members | SocketTools Namespace

NewsFeed Members

NewsFeed overview

Public Static (Shared) Fields

feedPortDefault	A constant value which specifies the default port number.
feedPortSecure	A constant value which specifies the default port number for a secure connection.
feedTimeout	A constant value which specifies the default timeout period.

Public Instance Constructors

NewsFeed Constructor	Initializes a new instance of the NewsFeed class.

Public Instance Properties

Category	Gets a value which specifies the category or categories that the channel belongs to.
Copyright	Gets a value which specifies a copyright notice for the content.
Description	Gets a value which describes the news feed channel.
Editor	Gets a value which identifies the person responsible for managing the content of the news feed.
FeedVersion	Gets a value which identifies the version of the news feed.
Generator	Gets a value which identifies the application that was used to create the news feed.
ImageLink	Gets a value which specifies a URL to the website corresponding to the news feed.
ImageTitle	Gets a value which describes the image associated with the news feed.
ImageUrl	Gets a value which specifies a URL for the image associated with the news feed.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
ItemAuthor	Gets a value which identifies the author of the current news feed item.
ItemComments	Gets a value which specifies a URL that links to further discussion about the current item.
ItemCount	Gets value which specifies the number of news items in the channel.

ItemEnclosure	Gets a value which specifies a URL that links to a file related to the item.
ItemGuid	Gets a value which uniquely identifies the current news item in the channel.
ItemId	Gets an numeric value which identifies the current news item in the channel.
ItemLink	Gets a value which specifies a URL that links to additional information related to the current item.
ItemPublished	Gets a value which specifies the date and time the current news item was published.
ItemSource	Gets a value which identifies the source of the current news item.
ItemText	Gets a value which provides a summary or description of the current news item.
ItemTitle	Gets a value which specifies a title for the current news item.
Language	Gets a value which identifies the language the news feed is written in.
LastBuild	Gets a value which specifies the date and time that the content of the news feed was last modified.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LinkUrl	Gets a value which specifies a URL to the website corresponding to the channel.
LocalFeed	Gets a value which specifies if the news feed was opened on the local system.
Options	Gets and sets a value which specifies one or more client options.
Published	Gets a value which specifies the date and time that the news feed was published.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
TimeToLive	Gets a value which specifies the frequency in seconds at which the feed should be refreshed.
Title	Gets a value which specifies the name of the news feed channel.
Trace	Gets and sets a value which indicates if network function logging is enabled.

TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
URL	Gets and sets a value which specifies the current news feed URL.
Version	Gets a value which returns the current version of the NewsFeed class library.
Webmaster	Gets a value which identifies the person responsible for technical issues related to the news feed.

Public Instance Methods

Close	Close the current news feed.
Dispose	Overloaded. Releases all resources used by NewsFeed.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
FindItem	Search for an item in the news feed channel which matches the unique identifier.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetItem	Set the current news item to the specified item number.
GetProperty	Overloaded. Get the value of a property for the specified item in the news feed.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the NewsFeed class.
Open	Overloaded. Open the specified news feed and load the first news item.
Parse	Overloaded. Parse the contents of a news feed and load the first news item.
Refresh	Refresh the current news feed, reloading the news channel items.
Reset	Reset the internal state of the object, resetting all properties to their default values.
Store	Store the contents of the news feed in an XML formatted text file.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any

	resources allocated for the current thread.

Public Instance Events

OnError	Occurs when an client operation fails.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the NewsFeed class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed Constructor

Initializes a new instance of the NewsFeed class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public NewsFeed();
```

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed Properties

The properties of the **NewsFeed** class are listed below. For a complete list of **NewsFeed** class members, see the NewsFeed Members topic.

Public Instance Properties

Category	Gets a value which specifies the category or categories that the channel belongs to.
Copyright	Gets a value which specifies a copyright notice for the content.
Description	Gets a value which describes the news feed channel.
Editor	Gets a value which identifies the person responsible for managing the content of the news feed.
FeedVersion	Gets a value which identifies the version of the news feed.
Generator	Gets a value which identifies the application that was used to create the news feed.
ImageLink	Gets a value which specifies a URL to the website corresponding to the news feed.
ImageTitle	Gets a value which describes the image associated with the news feed.
ImageUrl	Gets a value which specifies a URL for the image associated with the news feed.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
ItemAuthor	Gets a value which identifies the author of the current news feed item.
ItemComments	Gets a value which specifies a URL that links to further discussion about the current item.
ItemCount	Gets value which specifies the number of news items in the channel.
ItemEnclosure	Gets a value which specifies a URL that links to a file related to the item.
ItemGuid	Gets a value which uniquely identifies the current news item in the channel.
ItemId	Gets an numeric value which identifies the current news item in the channel.
ItemLink	Gets a value which specifies a URL that links to additional information related to the current item.
ItemPublished	Gets a value which specifies the date and time the current news item was published.

ItemSource	Gets a value which identifies the source of the current news item.
ItemText	Gets a value which provides a summary or description of the current news item.
ItemTitle	Gets a value which specifies a title for the current news item.
Language	Gets a value which identifies the language the news feed is written in.
LastBuild	Gets a value which specifies the date and time that the content of the news feed was last modified.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LinkUrl	Gets a value which specifies a URL to the website corresponding to the channel.
LocalFeed	Gets a value which specifies if the news feed was opened on the local system.
Options	Gets and sets a value which specifies one or more client options.
Published	Gets a value which specifies the date and time that the news feed was published.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
TimeToLive	Gets a value which specifies the frequency in seconds at which the feed should be refreshed.
Title	Gets a value which specifies the name of the news feed channel.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
URL	Gets and sets a value which specifies the current news feed URL.
Version	Gets a value which returns the current version of the NewsFeed class library.
Webmaster	Gets a value which identifies the person

	responsible for technical issues related to the news feed.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.Category Property

Gets a value which specifies the category or categories that the channel belongs to.

[Visual Basic]

```
Public ReadOnly Property Category As String
```

[C#]

```
public string Category {get;}
```

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.Copyright Property

Gets a value which specifies a copyright notice for the content.

[Visual Basic]

```
Public ReadOnly Property Copyright As String
```

[C#]

```
public string Copyright {get;}
```

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.Description Property

Gets a value which describes the news feed channel.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.Editor Property

Gets a value which identifies the person responsible for managing the content of the news feed.

[Visual Basic]

```
Public ReadOnly Property Editor As String
```

[C#]

```
public string Editor {get;}
```

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.FeedVersion Property

Gets a value which identifies the version of the news feed.

[Visual Basic]

```
Public ReadOnly Property FeedVersion As String
```

[C#]

```
public string FeedVersion {get;}
```

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.Generator Property

Gets a value which identifies the application that was used to create the news feed.

[Visual Basic]
```
Public ReadOnly Property Generator As String
```

[C#]
```
public string Generator {get;}
```

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.ImageLink Property

Gets a value which specifies a URL to the website corresponding to the news feed.

[Visual Basic]

```
Public ReadOnly Property ImageLink As String
```

[C#]

```
public string ImageLink {get;}
```

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.ImageTitle Property

Gets a value which describes the image associated with the news feed.

[Visual Basic]

```
Public ReadOnly Property ImageTitle As String
```

[C#]

```
public string ImageTitle {get;}
```

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.ImageUrl Property

Gets a value which specifies a URL for the image associated with the news feed.

[Visual Basic]

```
Public ReadOnly Property ImageUrl As String
```

[C#]

```
public string ImageUrl {get;}
```

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.ItemAuthor Property

Gets a value which identifies the author of the current news feed item.

[Visual Basic]

```
Public ReadOnly Property ItemAuthor As String
```

[C#]

```
public string ItemAuthor {get;}
```

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.ItemComments Property

Gets a value which specifies a URL that links to further discussion about the current item.

[Visual Basic]

```
Public ReadOnly Property ItemComments As String
```

[C#]

```
public string ItemComments {get;}
```

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.ItemCount Property

Gets value which specifies the number of news items in the channel.

[Visual Basic]
```
Public ReadOnly Property ItemCount As Integer
```

[C#]
```
public int ItemCount {get;}
```

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.ItemEnclosure Property

Gets a value which specifies a URL that links to a file related to the item.

[Visual Basic]

```
Public ReadOnly Property ItemEnclosure As String
```

[C#]

```
public string ItemEnclosure {get;}
```

Remarks

This is similar to an attachment in an email message, however instead of the item containing the contents of the attached file, it only specifies a link to the file. Enclosures are most commonly used with podcasting where an item is linked to an audio or video file, however the link may reference any type of file. If there is no enclosure specified for the current item, this property will return an empty string.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.ItemGuid Property

Gets a value which uniquely identifies the current news item in the channel.

[Visual Basic]

```
Public ReadOnly Property ItemGuid As String
```

[C#]

```
public string ItemGuid {get;}
```

Remarks

If this property is defined, it is guaranteed to be a unique, persistent value. It is important to note that this string does not have to be a standard GUID reference number, it can be any unique string. In many cases it is the same value as the hyperlink returned by the **ItemLink** property, although an application should never depend on this behavior. If there is no unique identifier associated with the current item, this property will return an empty string.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.ItemId Property

Gets an numeric value which identifies the current news item in the channel.

[Visual Basic]

```
Public ReadOnly Property ItemId As Integer
```

[C#]

```
public int ItemId {get;}
```

Remarks

The value returned by this property is an index into the list of available news items. It is not persistent and the ID for a specific news item may change when the news feed is refreshed or opened at a later point. To uniquely identify a news item in the channel, use the **ItemGuid** property.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.ItemLink Property

Gets a value which specifies a URL that links to additional information related to the current item.

[Visual Basic]
```
Public ReadOnly Property ItemLink As String
```

[C#]
```
public string ItemLink {get;}
```

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.ItemPublished Property

Gets a value which specifies the date and time the current news item was published.

[Visual Basic]

```
Public ReadOnly Property ItemPublished As String
```

[C#]

```
public string ItemPublished {get;}
```

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.ItemSource Property

Gets a value which identifies the source of the current news item.

[Visual Basic]

```
Public ReadOnly Property ItemSource As String
```

[C#]

```
public string ItemSource {get;}
```

Remarks

If the news item specifies a source, it will be the URL for the original news feed that contained it. This is typically used to propagate credit for news items that are aggregated by a third-party and re-published in their own channel. If the source is not specified, this property will return an empty string.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.ItemText Property

Gets a value which provides a summary or description of the current news item.

[Visual Basic]

```
Public ReadOnly Property ItemText As String
```

[C#]

```
public string ItemText {get;}
```

Remarks

This may property may return either plain text or HTML formatted text. If no text has been specified for the current item, this property will return an empty string.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.ItemTitle Property

Gets a value which specifies a title for the current news item.

[Visual Basic]

```
Public ReadOnly Property ItemTitle As String
```

[C#]

```
public string ItemTitle {get;}
```

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.Language Property

Gets a value which identifies the language the news feed is written in.

[Visual Basic]

```
Public ReadOnly Property Language As String
```

[C#]

```
public string Language {get;}
```

Remarks

This property typically returns standardized language codes, however the value actually returned depends on the content of the feed. If the news feed does not define this property, then it is generally presumed to be written in English.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.LastBuild Property

Gets a value which specifies the date and time that the content of the news feed was last modified.

[Visual Basic]

```
Public ReadOnly Property LastBuild As String
```

[C#]

```
public string LastBuild {get;}
```

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public NewsFeed.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.LinkUrl Property

Gets a value which specifies a URL to the website corresponding to the channel.

[Visual Basic]

```
Public ReadOnly Property LinkUrl As String
```

[C#]

```
public string LinkUrl {get;}
```

Remarks

This property does not return the URL of the news feed itself. Typically it is a link to the home page of the site which owns the news feed. If a link has not been specified in the news feed, this property will return an empty string. Note that strictly conforming news feeds require a valid link.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.LocalFeed Property

Gets a value which specifies if the news feed was opened on the local system.

[Visual Basic]

```
Public ReadOnly Property LocalFeed As Boolean
```

[C#]

```
public bool LocalFeed {get;}
```

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.Options Property

Gets and sets a value which specifies one or more client options.

[Visual Basic]
```
Public Property Options As NewsFeedOptions
```

[C#]
```
public NewsFeed.NewsFeedOptions Options {get; set;}
```

Property Value

Returns one or more NewsFeedOptions enumeration flags which specify the options for the client. The default value for this property is **optionNone**.

Remarks

The **Options** property specifies one or more default options options which are used when opening a news feed using the **Open** method.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.Published Property

Gets a value which specifies the date and time that the news feed was published.

[Visual Basic]

```
Public ReadOnly Property Published As String
```

[C#]

```
public string Published {get;}
```

Remarks

The value returned by this property is not necessarily the date that the news feed was last modified. For example, a feed that is associated with a weekly print publication may update this value once per week. If a publish date has not been specified for the feed, this property will return an empty string.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]
```
Public Property ThrowError As Boolean
```

[C#]
```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.Timeout Property

Gets and sets a value which specifies a timeout period in seconds.

[Visual Basic]

```
Public Property Timeout As Integer
```

[C#]

```
public int Timeout {get; set;}
```

Property Value

An integer value which specifies a timeout period in seconds.

Remarks

Setting the **Timeout** property specifies the number of seconds until a blocking operation fails and returns an error.

The timeout period is only used when the client is in blocking mode. Although this property can be changed when the client is in non-blocking mode, the value will be ignored until the client is returned to blocking mode.

For most applications it is recommended the timeout period be set between 10 and 20 seconds.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.TimeToLive Property

Gets a value which specifies the frequency in seconds at which the feed should be refreshed.

[Visual Basic]

```
Public ReadOnly Property TimeToLive As Integer
```

[C#]

```
public int TimeToLive {get;}
```

Property Value

An integer value that specifies the frequency in seconds at which the feed should be refreshed to obtain updated information. Not all feeds specify a time-to-live, in which case this member will have a value of zero.

Remarks

The value of the **TimeToLive** property should be considered advisory, and not all news feeds will provide this value. If the news feed does provide this value, it is recommended that you consider it to be the minimum interval at which you will poll the site for updates to the feed.

To update the contents of the current feed, use the **Refresh** method.

See Also

NewsFeed Class | SocketTools Namespace | Refresh Method

NewsFeed.Title Property

Gets a value which specifies the name of the news feed channel.

[Visual Basic]

```
Public ReadOnly Property Title As String
```

[C#]

```
public string Title {get;}
```

Remarks

If the content of the news feed corresponds to a website, this is value returned by this property is typically the same as the title of the website. If a title has not been specified, this property will return an empty string. Note that a strictly conforming news feed requires a title.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.Trace Property

Gets and sets a value which indicates if network function logging is enabled.

[Visual Basic]
```
Public Property Trace As Boolean
```

[C#]
```
public bool Trace {get; set;}
```

Property Value

Returns **true** if network function tracing is enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Trace** property is used to enable (or disable) the tracing of network function calls. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the SocketTools networking classes and is enabled or disabled for an entire process. This means that once trace logging is enabled for a given component, all of the function calls made by the process using any of the SocketTools classes will be logged. For example, if you have an application using both the File Transfer Protocol and Post Office Protocol classes, and you set the **Trace** property to **true**, function calls made by both classes will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all classes used by the process.

If trace logging is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the logfile is fairly large. Since logfiles can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

When redistributing your application, make sure that you include the **SocketTools10.TraceLog.dll** module with your installation. If this library is not present, then no trace output will be generated and the value of the **Trace** property will be ignored. Only those function calls made by the SocketTools networking classes will be logged. Calls made directly to the Windows Sockets API, or calls made by other classes, will not be logged.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.TraceFile Property

Gets and sets a value which specifies the name of the logfile.

[Visual Basic]

```
Public Property TraceFile As String
```

[C#]

```
public string TraceFile {get; set;}
```

Property Value

A string which specifies the name of the file.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named **SocketTools.log** is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFile** property should be set to the same value for each instance. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
 MyApp INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0 MyApp WRN:
connect(46, 192.0.0.1:1234, 16) returned -1 [10035] MyApp ERR: accept(46,
NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the control is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Note that if the specified file cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.TraceFlags Property

Gets and sets a value which specifies the client function tracing flags.

[Visual Basic]
```
Public Property TraceFlags As TraceOptions
```

[C#]
```
public NewsFeed.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration which specifies the amount of detail written to the trace logfile.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Because network function tracing is enabled per-process, the trace flags are shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFlags** property should be set to the same value for each instance. Changing the trace flags for any one instance of the class will affect the logging performed for all SocketTools classes used by the application.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being written and the error **errorOperationWouldBlock** occurs, a warning is generated because the application simply needs to attempt to write the data at a later time.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.URL Property

Gets and sets a value which specifies the current news feed URL.

[Visual Basic]

```
Public Property URL As String
```

[C#]

```
public string URL {get; set;}
```

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.Version Property

Gets a value which returns the current version of the NewsFeed class library.

[Visual Basic]

```
Public ReadOnly Property Version As String
```

[C#]

```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the NewsFeed class library. This value can be used by an application for validation and debugging purposes.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.Webmaster Property

Gets a value which identifies the person responsible for technical issues related to the news feed.

[Visual Basic]

```
Public ReadOnly Property Webmaster As String
```

[C#]

```
public string Webmaster {get;}
```

Remarks

If this value is defined in the news feed, it is typically the email address of a system administrator responsible for the server that hosts the news feed. If the webmaster is not specified, this property will return an empty string.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed Methods

The methods of the **NewsFeed** class are listed below. For a complete list of **NewsFeed** class members, see the NewsFeed Members topic.

Public Instance Methods

Close	Close the current news feed.
Dispose	Overloaded. Releases all resources used by NewsFeed.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
FindItem	Search for an item in the news feed channel which matches the unique identifier.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetItem	Set the current news item to the specified item number.
GetProperty	Overloaded. Get the value of a property for the specified item in the news feed.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the NewsFeed class.
Open	Overloaded. Open the specified news feed and load the first news item.
Parse	Overloaded. Parse the contents of a news feed and load the first news item.
Refresh	Refresh the current news feed, reloading the news channel items.
Reset	Reset the internal state of the object, resetting all properties to their default values.
Store	Store the contents of the news feed in an XML formatted text file.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the NewsFeed class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading

	the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.Close Method

Close the current news feed.

[Visual Basic]

```
Public Function Close() As Boolean
```

[C#]

```
public bool Close();
```

Return Value

A boolean value which specifies if the news feed was closed successfully. A return value of true indicates success, while a return value of false indicates failure. If the method fails, the value of the **LastError** property can be used to determine cause of the failure.

Remarks

The **Close** method must be called whenever the application has completed processing the news feed. It is important to note that information about the current news feed item will be cleared whenever this method is called, resetting the channel and item related properties back to their default values.

See Also

NewsFeed Class | SocketTools Namespace | Open Method | Parse Method

NewsFeed.FindItem Method

Search for an item in the news feed channel which matches the unique identifier.

```
[Visual Basic]
Public Function FindItem( _
   ByVal itemGuid As String _
) As Boolean
```

```
[C#]
public bool FindItem(
   string itemGuid
);
```

Parameters

itemGuid

A string value which specifies the unique identifier for the news item that is being searched for.

Return Value

A boolean value that specifies if the news item was found. A return value of true indicates success, while a return value of false indicates failure. If the method fails, the value of the **LastError** property can be used to determine cause of the failure.

Remarks

If this method returns true, the current news item is changed to the item that was found and property values such as **ItemLink** and **ItemText** will be updated. If this method returns false, the current news item is not changed.

It is recommended that you use this method with news feeds that are using version 2.0 or later of the RSS specification. If the feed uses an earlier version, items may not include a GUID property. It is also possible that a feed may omit the GUID property even though it is considered a requirement for the current RSS specification. For the broadest compatibility with all news feeds, an application should not depend on being able to search for a specific news feed item by its GUID.

See Also

NewsFeed Class | SocketTools Namespace | ItemGuid Property | GetItem Method

NewsFeed.GetItem Method

Set the current news item to the specified item number.

[Visual Basic]

```
Public Function GetItem( _
   ByVal itemId As Integer _
) As Boolean
```

[C#]

```
public bool GetItem(
   int itemId
);
```

Parameters

itemId

An integer value which specifies item in the news feed channel.

Return Value

A boolean value that specifies if the news item was found. A return value of true indicates success, while a return value of false indicates failure. If the method fails, the value of the **LastError** property can be used to determine cause of the failure.

Remarks

The item number is an index into the list of available news items in the current news feed. The first news item is one, and it increments for each additional item in the feed. If ***itemId*** parameter is zero or specifies a value larger than the number of items in the feed, this method will fail.

If this method returns true, the current news item is changed to the specified value and property values such as **ItemLink** and **ItemText** will be updated. If this method returns false, the current news item is not changed.

If this method fails, it typically indicates that the ***itemId*** parameter is invalid or that the feed does not contain any valid news items. The **ItemCount** property can be used to determine the number of items contained in the news feed channel.

See Also

NewsFeed Class | SocketTools Namespace | ItemCount Property | FindItem Method

NewsFeed.GetProperty Method

Get the value of a property for the specified item in the news feed.

Overload List

Get the value of a property for the specified item in the news feed.

public bool GetProperty(int,string,string,ref string);

Get the value of a property for the specified item in the news feed.

public bool GetProperty(int,string,ref string);

Get the value of a property for the specified item in the news feed.

public bool GetProperty(string,string,ref string);

Get the value of a property for the specified item in the news feed.

public bool GetProperty(string,ref string);

See Also

NewsFeed Class | SocketTools Namespace | GetItem

NewsFeed.GetProperty Method (Int32, String, String, String)

Get the value of a property for the specified item in the news feed.

[Visual Basic]

```
Overloads Public Function GetProperty( _
   ByVal itemId As Integer, _
   ByVal propName As String, _
   ByVal propAttribute As String, _
   ByRef propValue As String _
) As Boolean
```

[C#]

```
public bool GetProperty(
   int itemId,
   string propName,
   string propAttribute,
   ref string propValue
);
```

Parameters

itemId
An integer value which specifies the item in the news feed channel.

propName
A string value which specifies the property name.

propAttribute
A string value which specifies the property attribute.

propValue
A string value passed by reference that will contain the property value when the method returns.

Return Value

A boolean value that specifies if the property attribute was found. A return value of true indicates success, while a return value of false indicates failure. If the method fails, the value of the **LastError** property can be used to determine cause of the failure.

Remarks

The **GetProperty** method is primarily used with custom item properties that may be used with extensions to the news feed. The standard properties for an news feed item such as the title, link and description can be access using properties such as **ItemTitle**, **ItemLink** and **ItemText**. However, if items in the feed contain custom properties that are not part of the standard RSS format, this method can be used to obtain those values.

The item number is an index into the list of available news items in the current news feed. The first news item is one, and it increments for each additional item in the feed. If *itemId* parameter is zero or specifies a value larger than the number of items in the feed, this method will fail.

See Also

NewsFeed Class | SocketTools Namespace | NewsFeed.GetProperty Overload List | GetItem

NewsFeed.GetProperty Method (Int32, String, String)

Get the value of a property for the specified item in the news feed.

[Visual Basic]

```
Overloads Public Function GetProperty( _
   ByVal itemId As Integer, _
   ByVal propName As String, _
   ByRef propValue As String _
) As Boolean
```

[C#]

```
public bool GetProperty(
   int itemId,
   string propName,
   ref string propValue
);
```

See Also

NewsFeed Class | SocketTools Namespace | NewsFeed.GetProperty Overload List

NewsFeed.GetProperty Method (String, String, String)

Get the value of a property for the specified item in the news feed.

[Visual Basic]

```
Overloads Public Function GetProperty( _
   ByVal propName As String, _
   ByVal propAttribute As String, _
   ByRef propValue As String _
) As Boolean
```

[C#]

```
public bool GetProperty(
   string propName,
   string propAttribute,
   ref string propValue
);
```

See Also

NewsFeed Class | SocketTools Namespace | NewsFeed.GetProperty Overload List

NewsFeed.GetProperty Method (String, String)

Get the value of a property for the specified item in the news feed.

[Visual Basic]

```
Overloads Public Function GetProperty( _
   ByVal propName As String, _
   ByRef propValue As String _
) As Boolean
```

[C#]

```
public bool GetProperty(
   string propName,
   ref string propValue
);
```

See Also

NewsFeed Class | SocketTools Namespace | NewsFeed.GetProperty Overload List

NewsFeed.Initialize Method

Initialize an instance of the NewsFeed class.

Overload List

Initialize an instance of the NewsFeed class.

public bool Initialize();

Initialize an instance of the NewsFeed class.

public bool Initialize(string);

See Also

NewsFeed Class | SocketTools Namespace | Uninitialize Method

NewsFeed.Initialize Method ()

Initialize an instance of the NewsFeed class.

[Visual Basic]
```
Overloads Public Function Initialize() As Boolean
```

[C#]
```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the NewsFeed class, loading the networking library and allocating resources for the current thread. Typically it is not necessary to explicitly call this method because the instance of the class is initialized by the class constructor. However, if the **Uninitialize** method is called, the class must be re-initialized before any other methods are called.

See Also

NewsFeed Class | SocketTools Namespace | NewsFeed.Initialize Overload List | Uninitialize Method

NewsFeed.Initialize Method (String)

Initialize an instance of the NewsFeed class.

[Visual Basic]
```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]
```
public bool Initialize(
   string licenseKey
);
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the NewsFeed class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the NewsFeed class.

Example

The following example shows how to use the Initialize method to initialize an instance of the class. This example assumes that the license key string has been defined in code.

```
SocketTools.NewsFeed newsFeed = new SocketTools.NewsFeed();

if (newsFeed.Initialize(strLicenseKey) == false)
{
    MessageBox.Show(newsFeed.LastErrorString, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    return;
}
```

```
Dim newsFeed As New SocketTools.NewsFeed

If newsFeed.Initialize(strLicenseKey) = False Then
    MsgBox(newsFeed.LastErrorString, vbIconExclamation)
    Exit Sub
End If
```

See Also

NewsFeed Class | SocketTools Namespace | NewsFeed.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

NewsFeed.Open Method

Open the specified news feed and load the first news item.

Overload List

Open the specified news feed and load the first news item.

public bool Open(string);

Open the specified news feed and load the first news item.

public bool Open(string,int);

Open the specified news feed and load the first news item.

public bool Open(string,int,NewsFeedOptions);

See Also

NewsFeed Class | SocketTools Namespace | Parse Method

NewsFeed.Open Method (String, Int32, NewsFeedOptions)

Open the specified news feed and load the first news item.

[Visual Basic]

```
Overloads Public Function Open( _
   ByVal feedUrl As String, _
   ByVal timeout As Integer, _
   ByVal options As NewsFeedOptions _
) As Boolean
```

[C#]

```
public bool Open(
   string feedUrl,
   int timeout,
   NewsFeedOptions options
);
```

Parameters

feedUrl

A string value which specifies the URL for the news feed. To access a news feed on a web server, a standard http or https URL may be used. To access a file on the local system or network share, a file name or file URL may be specified.

timeout

The number of seconds that the client will wait for a response before failing the operation. This parameter is ignored if the ***feedUrl*** parameter specifies a local file name or URL.

options

One or more NewsFeedOptions enumeration flags which specify the options for the client. The default value for this property is **optionNone**.

Return Value

A boolean value that specifies if the news feed was opened. A return value of true indicates success, while a return value of false indicates failure. If the method fails, the value of the **LastError** property can be used to determine cause of the failure.

Remarks

A news feed may be local or remote, depending on the URL that is specified. If a local file name or file URL is specified for the feed, then it is opened locally and no network access is required. If an http or https URL is specified, then the **Open** method will attempt to download the feed from the remote host and store it temporarily on the local system. Accessing a remote feed requires that the application has permission to establish a connection with the remote host and will cause the application to block until the feed has been downloaded, the operation times out or an error occurs.

Although the **Open** method will meet the needs of most applications, if you require more complex functionality such as retrieving the feed asynchronously in the background or event notifications for large transfers, you can use the **SocketTools.HttpClient** class to download the news feed and then use the **Parse** method to parse the contents.

See Also

NewsFeed Class | SocketTools Namespace | NewsFeed.Open Overload List | Parse Method

NewsFeed.Open Method (String, Int32)

Open the specified news feed and load the first news item.

[Visual Basic]

```
Overloads Public Function Open( _
   ByVal feedUrl As String, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Open(
   string feedUrl,
   int timeout
);
```

Parameters

feedUrl

A string value which specifies the URL for the news feed. To access a news feed on a web server, a standard http or https URL may be used. To access a file on the local system or network share, a file name or file URL may be specified.

timeout

The number of seconds that the client will wait for a response before failing the operation. This parameter is ignored if the ***feedUrl*** parameter specifies a local file name or URL.

Return Value

A boolean value that specifies if the news feed was opened. A return value of true indicates success, while a return value of false indicates failure. If the method fails, the value of the **LastError** property can be used to determine cause of the failure.

Remarks

A news feed may be local or remote, depending on the URL that is specified. If a local file name or file URL is specified for the feed, then it is opened locally and no network access is required. If an http or https URL is specified, then the **Open** method will attempt to download the feed from the remote host and store it temporarily on the local system. Accessing a remote feed requires that the application has permission to establish a connection with the remote host and will cause the application to block until the feed has been downloaded, the operation times out or an error occurs.

Although the **Open** method will meet the needs of most applications, if you require more complex functionality such as retrieving the feed asynchronously in the background or event notifications for large transfers, you can use the **SocketTools.HttpClient** class to download the news feed and then use the **Parse** method to parse the contents.

See Also

NewsFeed Class | SocketTools Namespace | NewsFeed.Open Overload List | Parse Method

NewsFeed.Open Method (String)

Open the specified news feed and load the first news item.

[Visual Basic]

```
Overloads Public Function Open( _
   ByVal feedUrl As String _
) As Boolean
```

[C#]

```
public bool Open(
   string feedUrl
);
```

Parameters

feedUrl

A string value which specifies the URL for the news feed. To access a news feed on a web server, a standard http or https URL may be used. To access a file on the local system or network share, a file name or file URL may be specified.

Return Value

A boolean value that specifies if the news feed was opened. A return value of true indicates success, while a return value of false indicates failure. If the method fails, the value of the **LastError** property can be used to determine cause of the failure.

Remarks

A news feed may be local or remote, depending on the URL that is specified. If a local file name or file URL is specified for the feed, then it is opened locally and no network access is required. If an http or https URL is specified, then the **Open** method will attempt to download the feed from the remote host and store it temporarily on the local system. Accessing a remote feed requires that the application has permission to establish a connection with the remote host and will cause the application to block until the feed has been downloaded, the operation times out or an error occurs.

Although the **Open** method will meet the needs of most applications, if you require more complex functionality such as retrieving the feed asynchronously in the background or event notifications for large transfers, you can use the **SocketTools.HttpClient** class to download the news feed and then use the **Parse** method to parse the contents.

See Also

NewsFeed Class | SocketTools Namespace | NewsFeed.Open Overload List | Parse Method

NewsFeed.Parse Method

Parse the contents of a news feed and load the first news item.

Overload List

Parse the contents of a news feed and load the first news item.

public bool Parse(string);

Parse the contents of a news feed and load the first news item.

public bool Parse(string,NewsFeedOptions);

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.Parse Method (String, NewsFeedOptions)

Parse the contents of a news feed and load the first news item.

[Visual Basic]

```
Overloads Public Function Parse( _
   ByVal feedXml As String, _
   ByVal options As NewsFeedOptions _
) As Boolean
```

[C#]

```
public bool Parse(
   string feedXml,
   NewsFeedOptions options
);
```

Parameters

feedXml

A string which contains the contents of the news feed. The string must contain XML formatted data that conforms to the RSS standard specification and cannot specify an empty string.

options

One or more NewsFeedOptions enumeration flags which specify the options for the client. The default value for this property is **optionNone**.

Return Value

A boolean value that specifies if the contents of the ***feedXml*** parameter could be parsed. A return value of true indicates success, while a return value of false indicates failure. If the method fails, the value of the **LastError** property can be used to determine cause of the failure.

Remarks

The **Parse** method is an alternative to the **Open** method, enabling the application to process a news feed from alternative sources such as a database or compressed file. It is important to note that the string which contains the news feed XML must be properly formatted and conform to the RSS standard specification.

See Also

NewsFeed Class | SocketTools Namespace | NewsFeed.Parse Overload List

NewsFeed.Parse Method (String)

Parse the contents of a news feed and load the first news item.

[Visual Basic]

```
Overloads Public Function Parse( _
   ByVal feedXml As String _
) As Boolean
```

[C#]

```
public bool Parse(
   string feedXml
);
```

Parameters

feedXml

A string which contains the contents of the news feed. The string must contain XML formatted data that conforms to the RSS standard specification and cannot specify an empty string.

Return Value

A boolean value that specifies if the contents of the ***feedXml*** parameter could be parsed. A return value of true indicates success, while a return value of false indicates failure. If the method fails, the value of the **LastError** property can be used to determine cause of the failure.

Remarks

The **Parse** method is an alternative to the **Open** method, enabling the application to process a news feed from alternative sources such as a database or compressed file. It is important to note that the string which contains the news feed XML must be properly formatted and conform to the RSS standard specification.

See Also

NewsFeed Class | SocketTools Namespace | NewsFeed.Parse Overload List

NewsFeed.Refresh Method

Refresh the current news feed, reloading the news channel items.

[Visual Basic]
```
Public Function Refresh() As Boolean
```

[C#]
```
public bool Refresh();
```

Return Value

A boolean value that specifies if the news feed was refreshed. A return value of true indicates success, while a return value of false indicates failure. If the method fails, the value of the **LastError** property can be used to determine cause of the failure.

Remarks

When the **Refresh** method is called, the news feed is reloaded from the original source and the items in the channel are updated. For news feeds that are frequently updated, the **TimeToLive** property can provide a hint to the application as to how frequently the feed should be refreshed.

If the news feed was originally opened using an http or https URL, this function will download the updated feed from the remote host and store it temporarily on the local system. Accessing a remote feed requires that the application has permission to establish a connection with the remote host and will cause the application to block until the feed has been downloaded, the operation times out or an error occurs. The same timeout period and options will be used as when the feed was originally opened.

The **Refresh** method should only be used if the feed was opened using the **Open** method, otherwise the method will fail with an error indicating that the operation is not supported.

It is important that the application does not make any assumptions about the number of news items in the channel, or the content associated with those items after the **Refresh** method has been called. For example, never assume that the number of items in the news feed remains the same, or that the item IDs for each item remains the same. If you need to find a specific item in the news feed, use the **FindItem** method.

See Also

NewsFeed Class | SocketTools Namespace | FindItem Method | Open Method

NewsFeed.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. If a socket has been allocated, it will be released and any active connections will be terminated. All properties will be reset to their default values.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.Store Method

Store the contents of the news feed in an XML formatted text file.

[Visual Basic]

```
Public Function Store( _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool Store(
   string fileName
);
```

Parameters

fileName

A string value which specifies the name of the file on the local system. The contents of the news feed will be stored in this file. If the file does not exist, it will be created; otherwise it will overwrite the contents of the file.

Return Value

A boolean value that specifies if the news feed was stored on the local system or network. A return value of true indicates success, while a return value of false indicates failure. If the method fails, the value of the **LastError** property can be used to determine cause of the failure.

See Also

NewsFeed Class | SocketTools Namespace | Open Method | Parse Method

NewsFeed.Uninitialize Method

Uninitialize the class library and release any resources allocated for the current thread.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method terminates any active connection, releases resources allocated for the current thread and unloads the networking library. After this method has been called, no further client operations may be performed until the class instance has been re-initialized.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the class is no longer needed.

See Also

NewsFeed Class | SocketTools Namespace | Initialize Method

NewsFeed Events

The events of the **NewsFeed** class are listed below. For a complete list of **NewsFeed** class members, see the NewsFeed Members topic.

Public Instance Events

⚡ OnError	Occurs when an client operation fails.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.OnError Event

Occurs when an client operation fails.

[Visual Basic]
```
Public Event OnError As OnErrorEventHandler
```

[C#]
```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type NewsFeed.ErrorEventArgs containing data related to this event. The following **NewsFeed.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a client operation fails.

See Also

NewsFeed Class | SocketTools Namespace

NewsFeed.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see NewsFeed.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.NewsFeed.ErrorEventArgs

[Visual Basic]

```
Public Class NewsFeed.ErrorEventArgs
    Inherits EventArgs
```

[C#]

```
public class NewsFeed.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NewsFeed (in SocketTools.NewsFeed.dll)

See Also

NewsFeed.ErrorEventArgs Members | SocketTools Namespace

NewsFeed.ErrorEventArgs Members

NewsFeed.ErrorEventArgs overview

Public Instance Constructors

NewsFeed.ErrorEventArgs Constructor	Initializes a new instance of the NewsFeed.ErrorEventArgs class.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

NewsFeed.ErrorEventArgs Class | SocketTools Namespace

NewsFeed.ErrorEventArgs Constructor

Initializes a new instance of the NewsFeed.ErrorEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public NewsFeed.ErrorEventArgs();
```

See Also

NewsFeed.ErrorEventArgs Class | SocketTools Namespace

NewsFeed.ErrorEventArgs Properties

The properties of the **NewsFeed.ErrorEventArgs** class are listed below. For a complete list of **NewsFeed.ErrorEventArgs** class members, see the NewsFeed.ErrorEventArgs Members topic.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

NewsFeed.ErrorEventArgs Class | SocketTools Namespace

NewsFeed.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

NewsFeed.ErrorEventArgs Class | SocketTools Namespace | Error Property

NewsFeed.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public NewsFeed.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

NewsFeed.ErrorEventArgs Class | SocketTools Namespace | Description Property

NewsFeed.ErrorCode Enumeration

Specifies the error codes returned by the NewsFeed class.

[Visual Basic]
```
Public Enum NewsFeed.ErrorCode
```

[C#]
```
public enum NewsFeed.ErrorCode
```

Remarks

The NewsFeed class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.
errorProxyAuthenticationRequired	Proxy authentication required.

errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.
errorUnsupportedMediaType	Request specified an unsupported media type.

errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.
errorNoMoreMailboxes	No more mailboxes exist on this server.

errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.
errorInvalidThreadHandle	The specified thread handle is no longer valid.

errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been defined.

errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or unrecognized.

errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NewsFeed (in SocketTools.NewsFeed.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

NewsFeed.TraceOptions Enumeration

Specifies the logging options that the NewsFeed class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum NewsFeed.TraceOptions
```

[C#]
```
[Flags]
public enum NewsFeed.TraceOptions
```

Remarks

The NewsFeed class uses the **TraceOptions** enumeration to specify what kind of debugging information is written to the trace logfile. These options are only meaningful when trace logging is enabled by setting the **Trace** property to **true**.

Members

Member Name	Description	Value
traceDefault	The default trace logging option. This is the same as specifying the **traceInfo** option.	0
traceInfo	All network function calls are written to the trace file. This is the default value.	0
traceError	Only those network function calls which fail are recorded in the trace file.	1
traceWarning	Only those network function calls which fail, or return values which indicate a warning, are recorded in the trace file.	2
traceHexDump	All network function calls are written to the trace file, plus all the data that is sent or received is displayed, in both ASCII and hexadecimal format.	4
traceProcess	All function calls in the current process are logged, rather than only those functions in the current thread. This option is useful for multithreaded applications that are using worker threads.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.NewsFeed (in SocketTools.NewsFeed.dll)

See Also

SocketTools Namespace

NewsFeed.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub NewsFeed.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void NewsFeed.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender
The source of the event.

e
An ErrorEventArgs that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NewsFeed (in SocketTools.NewsFeed.dll)

See Also

SocketTools Namespace

NewsFeed.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see NewsFeed.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.NewsFeed.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class NewsFeed.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class NewsFeed.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the NewsFeed class.

Example

```
<Assembly: SocketTools.NewsFeed.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.NewsFeed.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.NewsFeed (in SocketTools.NewsFeed.dll)

See Also

NewsFeed.RuntimeLicenseAttribute Members | SocketTools Namespace

NewsFeed.RuntimeLicenseAttribute Members

NewsFeed.RuntimeLicenseAttribute overview

Public Instance Constructors

NewsFeed.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

NewsFeed.RuntimeLicenseAttribute Class | SocketTools Namespace

NewsFeed.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

```
[Visual Basic]
Public Sub New( _
   ByVal licenseKey As String _
)
```

```
[C#]
public NewsFeed.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the NewsFeed class.

See Also

NewsFeed.RuntimeLicenseAttribute Class | SocketTools Namespace

NewsFeed.RuntimeLicenseAttribute Properties

The properties of the **NewsFeed.RuntimeLicenseAttribute** class are listed below. For a complete list of **NewsFeed.RuntimeLicenseAttribute** class members, see the NewsFeed.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

NewsFeed.RuntimeLicenseAttribute Class | SocketTools Namespace

NewsFeed.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

NewsFeed.RuntimeLicenseAttribute Class | SocketTools Namespace

NewsFeedException Class

The exception that is thrown when a client error occurs.

For a list of all members of this type, see NewsFeedException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.NewsFeedException

[Visual Basic]

```
Public Class NewsFeedException
    Inherits ApplicationException
```

[C#]

```
public class NewsFeedException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A NewsFeedException is thrown by the NewsFeed class when an error occurs.

The default constructor for the NewsFeedException class sets the **ErrorCode** property to the last client error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.NewsFeed (in SocketTools.NewsFeed.dll)

See Also

NewsFeedException Members | SocketTools Namespace

NewsFeedException Members

NewsFeedException overview

Public Instance Constructors

NewsFeedException	Overloaded. Initializes a new instance of the NewsFeedException class.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

NewsFeedException Class | SocketTools Namespace

NewsFeedException Constructor

Initializes a new instance of the NewsFeedException class with the last network error code.

Overload List

Initializes a new instance of the NewsFeedException class with the last network error code.

```
public NewsFeedException();
```

Initializes a new instance of the NewsFeedException class with a specified error number.

```
public NewsFeedException(int);
```

Initializes a new instance of the NewsFeedException class with a specified error message.

```
public NewsFeedException(string);
```

Initializes a new instance of the NewsFeedException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
public NewsFeedException(string,Exception);
```

See Also

NewsFeedException Class | SocketTools Namespace

NewsFeedException Constructor ()

Initializes a new instance of the NewsFeedException class with the last network error code.

[Visual Basic]

```
Overloads Public Sub New()
```

[C#]

```
public NewsFeedException();
```

Remarks

The ctor constructor sets the **ErrorCode** property to the last client error that occurred. For more information about the errors that may occur, refer to the NewsFeed.ErrorCode enumeration.

See Also

NewsFeedException Class | SocketTools Namespace | NewsFeedException Constructor Overload List

NewsFeedException Constructor (String)

Initializes a new instance of the NewsFeedException class with a specified error message.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]

```
public NewsFeedException(
   string message
);
```

Parameters

message

The error message that explains the reason for the exception.

Remarks

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

NewsFeedException Class | SocketTools Namespace | NewsFeedException Constructor Overload List

NewsFeedException Constructor (String, Exception)

Initializes a new instance of the NewsFeedException class with a specified error message and a reference to the inner exception that is the cause of this exception.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal innerException As Exception _
)
```

[C#]

```
public NewsFeedException(
   string message,
   Exception innerException
);
```

Parameters

message
The error message that explains the reason for the exception.

innerException
The exception that is the cause of the current exception. If the ***innerException*** parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

NewsFeedException Class | SocketTools Namespace | NewsFeedException Constructor Overload List

NewsFeedException Constructor (Int32)

Initializes a new instance of the NewsFeedException class with a specified error number.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

[C#]

```
public NewsFeedException(
   int code
);
```

Parameters

code

An integer value which specifies an error code.

Remarks

This constructor sets the **ErrorCode** property to the specified error code. For more information about the errors that may occur, refer to the NewsFeed.ErrorCode enumeration.

See Also

NewsFeedException Class | SocketTools Namespace | NewsFeedException Constructor Overload List

NewsFeedException Properties

The properties of the **NewsFeedException** class are listed below. For a complete list of **NewsFeedException** class members, see the NewsFeedException Members topic.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

NewsFeedException Class | SocketTools Namespace

NewsFeedException.ErrorCode Property

Gets a value which specifies the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property ErrorCode As ErrorCode
```

[C#]

```
public NewsFeed.ErrorCode ErrorCode {get;}
```

Property Value

Returns a NewsFeed.ErrorCode enumeration value which specifies the error code.

Remarks

The **ErrorCode** property returns the error code that specifies the cause of the exception.

The default constructor for the NewsFeedException class sets the error code to the last network error that occurred. For more information about the errors that may occur, refer to the NewsFeed.ErrorCode enumeration.

See Also

NewsFeedException Class | SocketTools Namespace

NewsFeedException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]

```
Overrides Public ReadOnly Property Message As String
```

[C#]

```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

NewsFeedException Class | SocketTools Namespace

NewsFeedException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception. This value is the same as the values returned by the Windows Sockets API. For more information about socket error codes, see the Windows Socket Version 2 API error code documentation in MSDN.

See Also

NewsFeedException Class | SocketTools Namespace

NewsFeedException Methods

The methods of the **NewsFeedException** class are listed below. For a complete list of **NewsFeedException** class members, see the NewsFeedException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

NewsFeedException Class | SocketTools Namespace

NewsFeedException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]

```
Overrides Public Function ToString() As String
```

[C#]

```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

NewsFeedException Class | SocketTools Namespace

PopClient Class

Implements the Post Office Protocol.

For a list of all members of this type, see PopClient Members.

System.Object
 SocketTools.PopClient

[Visual Basic]

```
Public Class PopClient
    Implements IDisposable
```

[C#]

```
public class PopClient : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The Post Office Protocol (POP3) provides access to a user's new email messages on a mail server. Methods are provided for listing available messages and then retrieving those messages, storing them either in files or in memory. Once a user's messages have been downloaded to the local system, they are typically removed from the server. This is the most popular email protocol used by Internet Service Providers (ISPs) and the PopClient class provides a complete interface for managing a user's mailbox. This class is typically used in conjunction with the SocketTools.MailMessage class, which is used to process the messages that are retrieved from the server.

This class supports secure connections using the standard SSL and TLS protocols. Both implicit and explicit SSL connections can be established, enabling the class to work with a wide variety of servers.

Requirements

Namespace: SocketTools

Assembly: SocketTools.PopClient (in SocketTools.PopClient.dll)

See Also

PopClient Members | SocketTools Namespace

PopClient Members

PopClient overview

Public Static (Shared) Fields

popPortDefault	A constant value which specifies the default port number.
popPortSecure	A constant value which specifies the default port number for a secure connection.
popTimeout	A constant value which specifies the default timeout period.

Public Instance Constructors

PopClient Constructor	Initializes a new instance of the PopClient class.

Public Instance Properties

Authentication	Gets and sets the method used to authenticate the user.
AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
BearerToken	Gets and sets the bearer token used with OAuth 2.0 authentication.
Blocking	Gets and sets a value which indicates if the client is in blocking mode.
CertificateExpires	Get a value that specifies the date that the security certificate expires.
CertificateIssued	Get a value that specifies the date that the security certificate was issued.
CertificateIssuer	Get a value that provides information about the organization that issued the certificate.
CertificateName	Gets and sets a value that specifies the name of the client certificate.
CertificatePassword	Gets and sets the password associated with the client certificate.
CertificateStatus	Gets a value which indicates the status of the security certificate returned by the remote host.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateSubject	Gets a value that provides information about the organization that the server certificate was issued to.
CertificateUser	Gets and sets the user that owns the client certificate.

CipherStrength	Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.
Handle	Gets a value that specifies the client handle allocated for the current session.
HashStrength	Gets a value which specifies the length of the message digest that was selected for a secure connection.
HeaderField	Gets and sets the current header field name.
HeaderValue	Gets the value of the current header field.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsConnected	Gets a value which indicates if a connection to the remote host has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsReadable	Gets a value which indicates if there is data available to be read from the socket connection to the server.
IsWritable	Gets a value which indicates if data can be written to the client without blocking.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LastMessage	Gets the number of the last message available in the current mailbox.
LocalAddress	Gets the local Internet address that the client is bound to.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
MailboxSize	Gets the size of the current mailbox.
Message	Gets and sets the current message number.
MessageCount	Gets the number of messages available in the mailbox.
MessageFrom	Gets the address of the user who sent the message.

MessageSize	Gets the size of the current message in bytes.
MessageUID	Gets the UID for the current message.
Options	Gets and sets a value which specifies one or more client options.
Password	Gets and sets the password used to authenticate the client.
RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.
ResultCode	Gets a value which specifies the result of the last command executed by the server.
ResultString	Gets a string value which describes the result of the previous command.
Secure	Gets and sets a value which specifies if a secure connection is established.
SecureCipher	Gets a value that specifies the encryption algorithm used for a secure connection.
SecureHash	Gets a value that specifies the message digest algorithm used for a secure connection.
SecureKeyExchange	Gets a value that specifies the key exchange algorithm used for a secure connection.
SecureProtocol	Gets and sets a value which specifies the protocol used for a secure connection.
Status	Gets a value which specifies the current status of the client.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
UserName	Gets and sets the username used to authenticate the client session.
Version	Gets a value which returns the current version of

	the PopClient class library.

Public Instance Methods

AttachThread	Attach an instance of the class to the current thread
Cancel	Cancel the current blocking client operation.
ChangePassword	Change the mailbox password for the current user.
CloseMessage	Closes the current message.
Command	Overloaded. Send a custom command to the server.
Connect	Overloaded. Establish a connection with a remote host.
DeleteMessage	Overloaded. Flags a message for deletion from the current mailbox.
Disconnect	Terminate the connection with a remote host.
Dispose	Overloaded. Releases all resources used by PopClient.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetHeader	Overloaded. Returns the value of a header field from the specified message.
GetHeaders	Overloaded. Retrieves the headers for the specified message from the server.
GetMessage	Overloaded. Retrieve a message from the server and return the contents in a byte array.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the PopClient class.
OpenMessage	Overloaded. Open the specified message for reading.
Read	Overloaded. Read data from the server and store it in a byte array.
Reset	Reset the internal state of the object, resetting all properties to their default values.
SendMessage	Overloaded. Submits a message to the mail server for delivery.
StoreMessage	Overloaded. Retrieve a message from the current mailbox and store it in a file on the local system.
ToString (inherited from Object)	Returns a String that represents the current Object.

Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
Write	Overloaded. Write one or more bytes of data to the server.

Public Instance Events

OnCancel	Occurs when a blocking client operation is canceled.
OnConnect	Occurs when a connection is established with the remote host.
OnDisconnect	Occurs when the remote host disconnects from the local system.
OnError	Occurs when an client operation fails.
OnProgress	Occurs as a data stream is being read or written to the client.
OnRead	Occurs when data is available to be read from the client.
OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
OnWrite	Occurs when data can be written to the client.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the PopClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

PopClient Class | SocketTools Namespace

PopClient Constructor

Initializes a new instance of the PopClient class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public PopClient();
```

See Also

PopClient Class | SocketTools Namespace

PopClient Properties

The properties of the **PopClient** class are listed below. For a complete list of **PopClient** class members, see the PopClient Members topic.

Public Instance Properties

Authentication	Gets and sets the method used to authenticate the user.
AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
BearerToken	Gets and sets the bearer token used with OAuth 2.0 authentication.
Blocking	Gets and sets a value which indicates if the client is in blocking mode.
CertificateExpires	Get a value that specifies the date that the security certificate expires.
CertificateIssued	Get a value that specifies the date that the security certificate was issued.
CertificateIssuer	Get a value that provides information about the organization that issued the certificate.
CertificateName	Gets and sets a value that specifies the name of the client certificate.
CertificatePassword	Gets and sets the password associated with the client certificate.
CertificateStatus	Gets a value which indicates the status of the security certificate returned by the remote host.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateSubject	Gets a value that provides information about the organization that the server certificate was issued to.
CertificateUser	Gets and sets the user that owns the client certificate.
CipherStrength	Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.
Handle	Gets a value that specifies the client handle allocated for the current session.
HashStrength	Gets a value which specifies the length of the message digest that was selected for a secure connection.
HeaderField	Gets and sets the current header field name.
HeaderValue	Gets the value of the current header field.

HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsConnected	Gets a value which indicates if a connection to the remote host has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsReadable	Gets a value which indicates if there is data available to be read from the socket connection to the server.
IsWritable	Gets a value which indicates if data can be written to the client without blocking.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LastMessage	Gets the number of the last message available in the current mailbox.
LocalAddress	Gets the local Internet address that the client is bound to.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
MailboxSize	Gets the size of the current mailbox.
Message	Gets and sets the current message number.
MessageCount	Gets the number of messages available in the mailbox.
MessageFrom	Gets the address of the user who sent the message.
MessageSize	Gets the size of the current message in bytes.
MessageUID	Gets the UID for the current message.
Options	Gets and sets a value which specifies one or more client options.
Password	Gets and sets the password used to authenticate the client.
RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.

ResultCode	Gets a value which specifies the result of the last command executed by the server.
ResultString	Gets a string value which describes the result of the previous command.
Secure	Gets and sets a value which specifies if a secure connection is established.
SecureCipher	Gets a value that specifies the encryption algorithm used for a secure connection.
SecureHash	Gets a value that specifies the message digest algorithm used for a secure connection.
SecureKeyExchange	Gets a value that specifies the key exchange algorithm used for a secure connection.
SecureProtocol	Gets and sets a value which specifies the protocol used for a secure connection.
Status	Gets a value which specifies the current status of the client.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
UserName	Gets and sets the username used to authenticate the client session.
Version	Gets a value which returns the current version of the PopClient class library.

See Also

PopClient Class | SocketTools Namespace

PopClient.Authentication Property

Gets and sets the method used to authenticate the user.

[Visual Basic]
```
Public Property Authentication As PopAuthentication
```

[C#]
```
public PopClient.PopAuthentication Authentication {get; set;}
```

Property Value

A PopAuthentication enumeration value which specifies the authentication method.

Remarks

The **authLogin** and **authPlain** authentication methods require the mail server support the Simple Authentication and Security Layer (SASL) AUTH command as defined in RFC 5034. Most modern mail servers do support one or both of these methods, and they are generally preferred over the **authPassword** method when possible. However, for backwards compatibility with legacy servers, the class will default to using **authPassword** for client authentication.

The **authXOAuth2** and **authBearer** authentication methods are similar, but they are not interchangeable. Both use an OAuth 2.0 bearer token to authenticate the client session, but they differ in how the token is presented to the server. It is currently preferable to use the XOAUTH2 method because it is more widely available and some service providers do not yet support the OAUTHBEARER method.

See Also

PopClient Class | SocketTools Namespace | BearerToken Poperty | Password Property | UserName Property

PopClient.AutoResolve Property

Gets and sets a value that determines if host names and addresses are automatically resolved.

[Visual Basic]

```
Public Property AutoResolve As Boolean
```

[C#]

```
public bool AutoResolve {get; set;}
```

Property Value

Returns **true** if host names are automatically resolved to Internet addresses. The default value is **false**.

Remarks

Setting the **AutoResolve** property determines if the class automatically resolves host names and addresses specified by the **HostName** and **HostAddress** properties. If set to **true**, setting the **HostName** property will cause the class to automatically determine the corresponding IP address and update the **HostAddress** property accordingly. Likewise, setting the **HostAddress** property will cause the class to determine the host name and update the **HostName** property. Setting this property to **false** prevents the class from resolving host names until a connection attempt is made.

It is important to note that setting the **HostName** or **HostAddress** property may cause the current thread to block, sometimes for several seconds, until the name or address is resolved. To prevent this behavior, set this property value to **false**.

See Also

PopClient Class | SocketTools Namespace

PopClient.BearerToken Property

Gets and sets the bearer token used with OAuth 2.0 authentication.

[Visual Basic]
```
Public Property BearerToken As String
```

[C#]
```
public string BearerToken {get; set;}
```

Property Value

Returns a string which contains the bearer token. Assigning a value to this property sets the curent authentication type to use OAuth 2.0 authentication and updates the bearer token.

Remarks

Assigning a value to the **BearerToken** property will automatically change the current authentication method to use OAuth 2.0 if necessary.

You should only use an OAuth 2.0 authentication method if you understand the process of how to request the access token. Obtaining a bearer token requires registering your application with the mail service provider (e.g.: Microsoft or Google), getting a unique client ID associated with your application and then requesting the bearer token using the appropriate scope for the service. Obtaining the initial token will typically involve interactive confirmation on the part of the user, requiring they grant permission to your application to access their mail account.

Your application should not store the bearer token for later use. They usually have a relatively short lifespan, typically about an hour, and are designed to be used with the current client session. You should specify offline access as part of the OAuth 2.0 scope, and store the refresh token provided by the service. The refresh token has a much onger validity period and can be used to obtain a new access token when needed.

If the current authentication method does not use OAuth 2.0, this property will return an empty string and you should use the **Password** property to obtain the current user password.

See Also

PopClient Class | SocketTools Namespace | Authentication Property | Password Property | UserName Property

PopClient.Blocking Property

Gets and sets a value which indicates if the client is in blocking mode.

[Visual Basic]

```
Public Property Blocking As Boolean
```

[C#]

```
public bool Blocking {get; set;}
```

Property Value

Returns **true** if the client is in blocking mode; otherwise it returns **false**. The default value is **true**.

Remarks

Setting the **Blocking** property determines if client operations complete synchronously or asynchronously. If set to **true**, then each client operation (such as sending or receiving data) will return when the operation has completed or timed-out. If set to **false**, client operations will return immediately. If the operation would result in the client blocking (such as attempting to read data when no data has been sent by the remote host), an error is generated.

It is important to note that certain events, such as **OnDisconnect, OnRead** and **OnWrite** are only fired if the client is in non-blocking mode.

See Also

PopClient Class | SocketTools Namespace

PopClient.CertificateExpires Property

Get a value that specifies the date that the security certificate expires.

[Visual Basic]

```
Public ReadOnly Property CertificateExpires As String
```

[C#]

```
public string CertificateExpires {get;}
```

Property Value

A string which specifies a date using the local date and time format.

Remarks

The **CertificateExpires** property returns a string that specifies the date and time that the security certificate expires. This property will return an empty string if a secure connection has not been established with the remote host.

See Also

PopClient Class | SocketTools Namespace

PopClient.CertificateIssued Property

Get a value that specifies the date that the security certificate was issued.

[Visual Basic]

```
Public ReadOnly Property CertificateIssued As String
```

[C#]

```
public string CertificateIssued {get;}
```

Property Value

A string which specifies a date using the local date and time format.

Remarks

The **CertificateIssued** property returns a string that specifies the date and time that the security certificate was issued. This property will return an empty string if a secure connection has not been established with the remote host.

See Also

PopClient Class | SocketTools Namespace

PopClient.CertificateIssuer Property

Get a value that provides information about the organization that issued the certificate.

[Visual Basic]

```
Public ReadOnly Property CertificateIssuer As String
```

[C#]

```
public string CertificateIssuer {get;}
```

Property Value

A string that contains a comma separated list of name value pairs.

Remarks

The **CertificateIssuer** property returns a string that contains information about the organization that issued the server certificate. The string value is a comma separated list of tagged name and value pairs. In the nomenclature of the X.500 standard, each of these pairs are called a relative distinguished name (RDN), and when concatenated together, forms the issuer's distinguished name (DN). For example:

```
C=US, O="RSA Data Security, Inc.", OU=Secure Server Certification Authority
```

To obtain a specific value, such as the name of the issuer or the issuer's country, the application must parse the string returned by this property. Some of the common tokens used in the distinguished name are:

Token	Description
C	The ISO standard two character country code.
S	The name of the state or province.
L	The name of the city or locality.
O	The name of the company or organization.
OU	The name of the department or organizational unit
CN	The common name; with X.509 certificates, this is the domain name of the site the certificate was issued for.

This property will return an empty string if a secure connection has not been established with the remote host.

See Also

PopClient Class | SocketTools Namespace

PopClient.CertificateName Property

Gets and sets a value that specifies the name of the client certificate.

[Visual Basic]

```
Public Property CertificateName As String
```

[C#]

```
public string CertificateName {get; set;}
```

Property Value

A string which specifies the certificate name.

Remarks

The **CertificateName** property is used to specify the name of a client certificate to use when establishing a secure connection. It is only required that you set this property value if the server requires a client certificate for authentication. If this property is not set, a client certificate will not be provided to the server. If a certificate name is specified, the certificate must have a private key associated with it, otherwise the connection attempt will fail because the control will be unable to create a security context for the session.

When the certificate store is searched for a matching certificate, it will first search for any certificate with a friendly name that matches the property value. If no valid certificate is found, it will then search for a certificate with a matching common name.

Certificates may be installed and viewed on the local system using the Certificate Manager that is included with the Windows operating system. For more information, refer to the documentation for the Microsoft Management Console.

See Also

PopClient Class | SocketTools Namespace | CertificateStore Property | Secure Property

PopClient.CertificateStatus Property

Gets a value which indicates the status of the security certificate returned by the remote host.

[Visual Basic]

```
Public ReadOnly Property CertificateStatus As SecurityCertificate
```

[C#]

```
public PopClient.SecurityCertificate CertificateStatus {get;}
```

Property Value

A SecurityCertificate enumeration value which specifies the status of the certificate.

Remarks

The **CertificateStatus** property is used to determine the status of the security certificate returned by the remote host when a secure connection has been established. This property value should be checked after the connection to the server has completed, but prior to beginning a transaction.

Note that if the certificate cannot be validated, the secure connection will not be automatically terminated. It is the responsibility of your application to determine the best course of action to take if the certificate is invalid. Even if the security certificate cannot be validated, the data exchanged with the remote host will still be encrypted.

See Also

PopClient Class | SocketTools Namespace

PopClient.CertificateStore Property

Gets and sets a value that specifies the name of the local certificate store.

[Visual Basic]
```
Public Property CertificateStore As String
```

[C#]
```
public string CertificateStore {get; set;}
```

Property Value

A string which specifies the certificate store name. The default value is the current user's personal certificate store.

Remarks

The **CertificateStore** property is used to specify the name of the certificate store which contains the security certificate to use when establishing a secure connection. The certificate may either be stored in the registry or in a file. If the certificate is stored in the registry, then this property should be set to one of the following predefined values:

Store Name	Description
CA	Certification authority certificates. These are certificates that are issued by entities which are entrusted to issue certificates to other individuals or organizations. Companies such as VeriSign and Thawte act as certification authorities.
MY	Personal certificates and their associated private keys for the current user. This store typically holds the client certificates used to establish a user's credentials. If a certificate store is not specified, this is the default value that is used.
ROOT	Certificates that have been self-signed by a certificate authority. Root certificates for a number of different certification authorities such as VeriSign and Thawte are installed as part of the operating system and periodically updated by Microsoft.

In most cases the client certificate will be installed in the user's personal certificate store, and therefore it is not necessary to set this property value because that is the default location that will be used to search for the certificate. This property is only used if the **CertificateName** property is also set to a valid certificate name.

If you are using a local certificate store, with the certificate and private key stored in the registry, you can explicitly specify whether the certificate store for the current user or the local machine (all users) should be used. This is done by prefixing the certificate store name with "HKCU" for the current user, or "HKLM" for the local machine. For example, a certificate store name of "HKLM:MY" would specify the personal certificate store for the local machine, rather than the current user. If neither prefix is specified, it will default to the certificate store for the current user.

This property may also be used to specify a file that contains the client certificate. In this case, the property should specify the full path to the file and must contain both the certificate and private key in PKCS12

format. If the file is protected by a password, the **CertificatePassword** property must also be set to specify the password.

See Also

PopClient Class | SocketTools Namespace | CertificatePassword Property | Secure Property

PopClient.CertificateSubject Property

Gets a value that provides information about the organization that the server certificate was issued to.

[Visual Basic]

```
Public ReadOnly Property CertificateSubject As String
```

[C#]

```
public string CertificateSubject {get;}
```

Property Value

A string that contains a comma separated list of name value pairs.

Remarks

The **CertificateSubject** property returns a string that contains information about the organization that the server certificate was issued to. The string value is a comma separated list of tagged name and value pairs. In the nomenclature of the X.500 standard, each of these pairs are called a relative distinguished name (RDN), and when concatenated together, forms the issuer's distinguished name (DN). For example:

```
C=US, O="RSA Data Security, Inc.", OU=Secure Server Certification Authority
```

To obtain a specific value, such as the name of the issuer or the issuer's country, the application must parse the string returned by this property. Some of the common tokens used in the distinguished name are:

Token	Description
C	The ISO standard two character country code.
S	The name of the state or province.
L	The name of the city or locality.
O	The name of the company or organization.
OU	The name of the department or organizational unit
CN	The common name; with X.509 certificates, this is the domain name of the site the certificate was issued for.

This property will return an empty string if a secure connection has not been established with the remote host.

See Also

PopClient Class | SocketTools Namespace

PopClient.CipherStrength Property

Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.

[Visual Basic]

```
Public ReadOnly Property CipherStrength As Integer
```

[C#]

```
public int CipherStrength {get;}
```

Property Value

An integer value which specifies the encryption key length if a secure connection has been established; otherwise a value of 0 is returned.

Remarks

The **CipherStrength** property returns the number of bits in the key used to encrypt the secure data stream. Common values returned by this property are 128 and 256. A key length of 40 or 56 bits is considered insecure and subject to brute force attacks. 128-bit and 256-bit keys are considered secure. If this property returns a value of 0, this means that a secure connection has not been established with the remote host.

See Also

PopClient Class | SocketTools Namespace

PopClient.Handle Property

Gets a value that specifies the client handle allocated for the current session.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer which represents a client handle. If there is no active connection, a value of -1 is returned.

Remarks

The **Handle** property specifies the numeric descriptor of the current client session.

See Also

PopClient Class | SocketTools Namespace

PopClient.HashStrength Property

Gets a value which specifies the length of the message digest that was selected for a secure connection.

[Visual Basic]

```
Public ReadOnly Property HashStrength As Integer
```

[C#]

```
public int HashStrength {get;}
```

Property Value

An integer value which specifies the length of the message digest if a secure connection has been established; otherwise a value of 0 is returned.

Remarks

The **HashStrength** property returns the number of bits used in the message digest (hash) that was selected. Common values returned by this property are 128 and 160. If this property returns a value of 0, this means that a secure connection has not been established with the remote host.

See Also

PopClient Class | SocketTools Namespace

PopClient.HeaderField Property

Gets and sets the current header field name.

[Visual Basic]
```
Public Property HeaderField As String
```

[C#]
```
public string HeaderField {get; set;}
```

Property Value

A string which specifies the current header field name.

Remarks

The **HeaderField** property returns the name of the current header field. Setting this property causes the control to determine if that header field exists, and if it does, to update the **HeaderValue** property with its value. This property can be used to obtain the value of any header in the current message.

See Also

PopClient Class | SocketTools Namespace

PopClient.HeaderValue Property

Gets the value of the current header field.

[Visual Basic]

```
Public ReadOnly Property HeaderValue As String
```

[C#]

```
public string HeaderValue {get;}
```

Property Value

A string which specifies the value of the current header field.

Remarks

The **HeaderValue** property returns the value of the header field specified by the **HeaderField** property. This property can be used to obtain the value of any header in the current message.

See Also

PopClient Class | SocketTools Namespace

PopClient.HostAddress Property

Gets and sets a value which specifies the Internet address used to establish a connection.

[Visual Basic]

```
Public Property HostAddress As String
```

[C#]

```
public string HostAddress {get; set;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **HostAddress** property can be used to set the Internet address for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the address is assigned to a valid host name, the **HostName** property will be updated with that value.

See Also

PopClient Class | SocketTools Namespace

PopClient.HostName Property

Gets and sets a value which specifies the host name used to establish a connection.

[Visual Basic]

```
Public Property HostName As String
```

[C#]

```
public string HostName {get; set;}
```

Property Value

A string which specifies a host name.

Remarks

The **HostName** property can be used to set the host name for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the name can be resolved to a valid Internet address, the **HostAddress** property will be updated with that value.

See Also

PopClient Class | SocketTools Namespace

PopClient.IsBlocked Property

Gets a value which indicates if the current thread is performing a blocking client operation.

[Visual Basic]
```
Public ReadOnly Property IsBlocked As Boolean
```

[C#]
```
public bool IsBlocked {get;}
```

Property Value

Returns **true** if the current thread is blocking, otherwise returns **false**.

Remarks

The **IsBlocked** property returns **true** if the current thread is blocked performing an operation. Because the Windows Sockets API only permits one blocking operation per thread of execution, this property should be checked before starting any blocking operation in response to an event.

If the **IsBlocked** property returns **false**, this means there are no blocking operations on the current thread at that time. However, this does not guarantee that the next client operation will not fail. An application should always check the return value from a client operation and check the value of the **LastError** property if an error occurs.

Note that this property will return **true** if there is any blocking operation being performed by the current thread, regardless of whether this specific instance of the class is responsible for the blocking operation or not.

See Also

PopClient Class | SocketTools Namespace

PopClient.IsConnected Property

Gets a value which indicates if a connection to the remote host has been established.

[Visual Basic]

```
Public ReadOnly Property IsConnected As Boolean
```

[C#]

```
public bool IsConnected {get;}
```

Property Value

Returns **true** if the connection has been established; otherwise returns **false**.

Remarks

The **IsConnected** property can only be used to indicate if there is still a logical connection to the remote host. It cannot be used to detect abnormal conditions such as the remote host aborting the connection, the physical network connection being lost or other critical errors.

See Also

PopClient Class | SocketTools Namespace

PopClient.IsInitialized Property

Gets a value which indicates if the current instance of the class has been initialized successfully.

[Visual Basic]
```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]
```
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the class instance has been initialized; otherwise returns **false**.

Remarks

The **IsInitialized** property is used to determine if the current instance of the class has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the class may not be able to initialize itself.

The most common reasons that a class instance may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required networking libraries or not being able to access the system registry.

See Also

PopClient Class | SocketTools Namespace

PopClient.IsReadable Property

Gets a value which indicates if there is data available to be read from the socket connection to the server.

[Visual Basic]
```
Public ReadOnly Property IsReadable As Boolean
```

[C#]
```
public bool IsReadable {get;}
```

Property Value

Returns **true** if there is data available to be read; otherwise returns **false**.

Remarks

The **IsReadable** property returns **true** if data can be read from the client without blocking. For non-blocking sessions, this property can be checked before the application attempts to read the client. Note that even if this property does return **true** indicating that there is data available to be read, applications should always check the return value from the **Read** method.

See Also

PopClient Class | SocketTools Namespace

PopClient.IsWritable Property

Gets a value which indicates if data can be written to the client without blocking.

[Visual Basic]

```
Public ReadOnly Property IsWritable As Boolean
```

[C#]

```
public bool IsWritable {get;}
```

Property Value

Returns **true** if data can be written to the client; otherwise returns **false**.

Remarks

The **IsWritable** property returns **true** if data can be written to the client without blocking. For non-blocking sessions, this property can be checked before the application attempts to write data to the client. Note that even if this property does return **true** indicating that data can be written to the client, applications should always check the return value from the **Write** method.

See Also

PopClient Class | SocketTools Namespace

PopClient.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public PopClient.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

PopClient Class | SocketTools Namespace

PopClient.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

PopClient Class | SocketTools Namespace

PopClient.LastMessage Property

Gets the number of the last message available in the current mailbox.

[Visual Basic]

```
Public ReadOnly Property LastMessage As Integer
```

[C#]

```
public int LastMessage {get;}
```

Property Value

An integer value which specifies the number of messages.

Remarks

The **LastMessage** property returns the number of the last message available in the currently selected mailbox. This value may be different than the value returned by the **MessageCount** property if one or more messages have been deleted. This is because the **MessageCount** property returns the number of available messages, and this value will decrement whenever a message is flagged for deletion. The **LastMessage** property value is constant and will always return the last message number in the mailbox, regardless if any messages have been deleted.

See Also

PopClient Class | SocketTools Namespace

PopClient.LocalAddress Property

Gets the local Internet address that the client is bound to.

[Visual Basic]

```
Public ReadOnly Property LocalAddress As String
```

[C#]

```
public string LocalAddress {get;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **LocalAddress** property returns the local Internet address that the client is bound to when a connection is established with a remote host. This property may return either an IPv4 or IPv6 formatted address, depending on the type of connection that was established.

See Also

PopClient Class | SocketTools Namespace

PopClient.LocalName Property

Gets a value which specifies the host name for the local system.

[Visual Basic]

```
Public ReadOnly Property LocalName As String
```

[C#]

```
public string LocalName {get;}
```

Property Value

A string which specifies the local host name.

Remarks

The **LocalName** property returns the fully-qualified host name assigned to the local system. If the system has not been configured with an Internet domain name, then this property will return the NetBIOS name assigned to the local system.

See Also

PopClient Class | SocketTools Namespace

PopClient.LocalPort Property

Gets the local port number the client is bound to.

[Visual Basic]

```
Public ReadOnly Property LocalPort As Integer
```

[C#]

```
public int LocalPort {get;}
```

Property Value

An integer value which specifies a port number. The default value is 0.

Remarks

The **LocalPort** property is used to identify the local port number that the client is bound to to when a connection is established with a remote host.

See Also

PopClient Class | SocketTools Namespace

PopClient.MailboxSize Property

Gets the size of the current mailbox.

[Visual Basic]

```
Public ReadOnly Property MailboxSize As Integer
```

[C#]

```
public int MailboxSize {get;}
```

Property Value

An integer value which specifies the size of the mailbox in bytes.

Remarks

The **MailboxSize** property returns the combined size in bytes of all of the available messages in the current mailbox. Note that as messages are deleted from the mailbox, this property value will decrease.

See Also

PopClient Class | SocketTools Namespace

PopClient.Message Property

Gets and sets the current message number.

[Visual Basic]

```
Public Property Message As Integer
```

[C#]

```
public int Message {get; set;}
```

Property Value

An integer value which specifies the current message number.

Remarks

The **Message** property sets or returns the message number for the currently selected mailbox. Message numbers range from 1 through the number of messages available on the server, as returned by the **MessageCount** property. Setting the **Message** property to an invalid message number will generate an exception.

See Also

PopClient Class | SocketTools Namespace

PopClient.MessageCount Property

Gets the number of messages available in the mailbox.

[Visual Basic]
```
Public ReadOnly Property MessageCount As Integer
```

[C#]
```
public int MessageCount {get;}
```

Property Value

An integer value which specifies the number of messages.

Remarks

The **MessageCount** property returns the number of messages available to be retrieved from the current mailbox.

See Also

PopClient Class | SocketTools Namespace

PopClient.MessageFrom Property

Gets the address of the user who sent the message.

[Visual Basic]
```
Public ReadOnly Property MessageFrom As String
```

[C#]
```
public string MessageFrom {get;}
```

Property Value

A string which specifies the email address of the user who sent the message.

Remarks

The **MessageFrom** property returns the address of the user who sent the current message. This property requires that the mail server support either the XSENDER or the XTND XLST command in order to determine who the sender is. The XSENDER command returns an authenticated address, as used with the Netscape SMTP authentication method. If this command is not supported, or the sender's address was not authenticated, then the XTND XLST command is used to return the value of the From header field in the message. If this command is not supported, the property will return an empty string.

See Also

PopClient Class | SocketTools Namespace

PopClient.MessageSize Property

Gets the size of the current message in bytes.

[Visual Basic]

```
Public ReadOnly Property MessageSize As Integer
```

[C#]

```
public int MessageSize {get;}
```

Property Value

An integer value which specifies the size of the message.

Remarks

The **MessageSize** property returns the size of the current message in bytes. The size includes the header and body portion of the message.

See Also

PopClient Class | SocketTools Namespace

PopClient.MessageUID Property

Gets the UID for the current message.

[Visual Basic]

```
Public ReadOnly Property MessageUID As String
```

[C#]

```
public string MessageUID {get;}
```

Property Value

An string value which specifies the current message UID.

Remarks

The **MessageID** property returns a string which uniquely identifies the message on the server. The identifier is assigned by the mail server, and retains the same value across multiple client sessions. This value is typically used when the client wants to leave a message on the mail server, but does not wish to retrieve the message contents multiple times. For example, the client can store the unique ID for each message that it retrieves, but does not delete from the server. The next time that it connects to the mail server, it compares the unique ID of a message against the stored values. If there is a match, the client knows that the message has already been retrieved, and does not need to do so again.

This property requires that the server support the optional UIDL command. If the command is not supported, this property will always return an empty string. Note that the unique ID for the message is not the same as the Message-ID header field in the message itself.

See Also

PopClient Class | SocketTools Namespace

PopClient.Options Property

Gets and sets a value which specifies one or more client options.

[Visual Basic]
```
Public Property Options As PopOptions
```

[C#]
```
public PopClient.PopOptions Options {get; set;}
```

Property Value

Returns one or more PopOptions enumeration flags which specify the options for the client. The default value for this property is **popOptionNone**.

Remarks

The **Options** property specifies one or more default options options which are used when establishing a connection using the **Connect** method.

See Also

PopClient Class | SocketTools Namespace

PopClient.Password Property

Gets and sets the password used to authenticate the client.

[Visual Basic]
```
Public Property Password As String
```

[C#]
```
public string Password {get; set;}
```

Property Value

A string which specifies the password.

Remarks

The **Password** property specifies the password used to authenticate the client session. This property is used as the default value for the **Connect** method if no password is specified as an argument.

Refer to the **Authentication** property for more information on the available authentication methods. If you are using the OAuth 2.0 authentication method, this property should not be set to the user's password. Instead, you should set the **BearerToken** property to the OAuth 2.0 bearer token issued by the mail service provider. Note that these access tokens can be much larger than your typical password and are only valid for a limited period of time.

You can use the **Password** property to specify an OAuth 2.0 bearer token. However, it is recommended that you use the **BearerToken** property instead of assigning it to this property. It will ensure compatibility with future versions of the class and make it clear in your code you are using an OAuth 2.0 bearer token and not a password. If the **AuthType** property specifies one of the OAuth 2.0 authentication methods, this property will return the bearer token.

See Also

PopClient Class | SocketTools Namespace | Authentication Property | BearerToken Property | UserName Property | Connect Method

PopClient.RemotePort Property

Gets and sets a value which specifies the remote port number.

[Visual Basic]

```
Public Property RemotePort As Integer
```

[C#]

```
public int RemotePort {get; set;}
```

Property Value

An integer value which specifies a port number.

Remarks

The **RemotePort** property is used to set the port number that will be used to establish a connection with a remote host. If the port number specifies a well-known port, the **RemoteService** property will be updated with that name.

See Also

PopClient Class | SocketTools Namespace

PopClient.ResultCode Property

Gets a value which specifies the result of the last command executed by the server.

[Visual Basic]
```
Public ReadOnly Property ResultCode As Boolean
```

[C#]
```
public bool ResultCode {get;}
```

Property Value

A boolean value which specifies the result of the last command executed on the server. A return value of **true** specifies that the previous command executed successfully. A return value of **false** specifies that the command failed.

Remarks

Unlike other protocols which typically return a numeric result code, the POP3 protocol returns a value that indicates only success or failure. If a failure occurs, the client should check the value of the **ResultString** property for any additional information as to what may have caused the failure.

See Also

PopClient Class | SocketTools Namespace

PopClient.ResultString Property

Gets a string value which describes the result of the previous command.

[Visual Basic]

```
Public ReadOnly Property ResultString As String
```

[C#]

```
public string ResultString {get;}
```

Property Value

A string which describes the result of the previous command executed on the server.

Remarks

The **ResultString** property returns the result string from the last action taken by the client. This string is generated by the remote server, and typically is used to describe the result code. For example, if an error is indicated by the result code, the result string may describe the condition that caused the error.

See Also

PopClient Class | SocketTools Namespace

PopClient.Secure Property

Gets and sets a value which specifies if a secure connection is established.

[Visual Basic]

```
Public Property Secure As Boolean
```

[C#]

```
public bool Secure {get; set;}
```

Property Value

Returns **true** if a secure connection is established; otherwise returns **false**. The default value is **false**.

Remarks

The **Secure** property determines if a secure connection is established with the remote host. The default value for this property is **false**, which specifies that a standard connection to the server is used. To establish a secure connection, the application should set this property value to **true** prior to calling the **Connect** method. Once the connection has been established, the client may exchange data with the server as with standard connections.

It is strongly recommended that any application that sets this property **true** use error handling to trap an errors that may occur. If the control is unable to initialize the security libraries, or otherwise cannot create a secure session for the client, an exception may be generated when this property value is set.

See Also

PopClient Class | SocketTools Namespace

PopClient.SecureCipher Property

Gets a value that specifies the encryption algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureCipher As SecureCipherAlgorithm
```

[C#]

```
public PopClient.SecureCipherAlgorithm SecureCipher {get;}
```

Property Value

A SecureCipherAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureCipher** property returns a value which identifies the algorithm used to encrypt the data stream. If a secure connection has not been established, this property will return a value of **cipherNone**.

See Also

PopClient Class | SocketTools Namespace

PopClient.SecureHash Property

Gets a value that specifies the message digest algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureHash As SecureHashAlgorithm
```

[C#]

```
public PopClient.SecureHashAlgorithm SecureHash {get;}
```

Property Value

A SecureHashAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureHash** property returns a value which identifies the message digest (hash) algorithm that was selected when a secure connection was established. If a secure connection has not been established, this property will return a value of **hashNone**.

See Also

PopClient Class | SocketTools Namespace

PopClient.SecureKeyExchange Property

Gets a value that specifies the key exchange algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureKeyExchange As SecureKeyAlgorithm
```

[C#]

```
public PopClient.SecureKeyAlgorithm SecureKeyExchange {get;}
```

Property Value

A SecureKeyAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureKeyExchange** property returns a value which identifies the key exchange algorithm that was selected when a secure connection was established. If a secure connection has not been established, this property will return a value of **keyExchangeNone**.

See Also

PopClient Class | SocketTools Namespace

PopClient.SecureProtocol Property

Gets and sets a value which specifies the protocol used for a secure connection.

[Visual Basic]
```
Public Property SecureProtocol As SecurityProtocols
```

[C#]
```
public PopClient.SecurityProtocols SecureProtocol {get; set;}
```

Property Value

A SecurityProtocols enumeration value which identifies the protocol to be used when establishing a secure connection.

Remarks

The **SecureProtocol** property can be used to specify the security protocol to be used when establishing a secure connection with a server or accepting a secure connection from a client. By default, the class will attempt to use either SSL v3 or TLS v1 to establish the connection, with the appropriate protocol automatically selected based on the capabilities of the remote host. It is recommended that you only change this property value if you fully understand the implications of doing so. Assigning a value to this property will override the default protocol and force the class to attempt to use only the protocol specified.

Multiple security protocols may be specified by combining them using a bitwise **or** operator. After a connection has been established, this property will identify the protocol that was selected. Attempting to set this property after a connection has been established will result in an exception being thrown. This property should only be set after setting the **Secure** property to **true** and before calling the **Accept** or **Connect** methods.

In some cases, a server may only accept a secure connection if the TLS v1 protocol is specified. If the security protocol is not compatible with the server, then the connection will fail with an error indicating that the control is unable to establish a security context for the session. In this case, try assigning the property to **protocolTLS1** and attempt the connection again.

See Also

PopClient Class | SocketTools Namespace

PopClient.Status Property

Gets a value which specifies the current status of the client.

[Visual Basic]

```
Public ReadOnly Property Status As PopStatus
```

[C#]

```
public PopClient.PopStatus Status {get;}
```

Property Value

A PopStatus enumeration value which specifies the current client status.

Remarks

The **Status** property returns the current status of the client. This property can be used to check on blocking connections to determine if the client is interacting with the remote host before taking some action.

See Also

PopClient Class | SocketTools Namespace

PopClient.ThreadModel Property

Gets and sets a value which specifies the threading model for the class instance.

[Visual Basic]
```
Public Property ThreadModel As ThreadingModel
```

[C#]
```
public PopClient.ThreadingModel ThreadModel {get; set;}
```

Property Value

Returns one or more ThreadingModel enumeration value which specifies the threading model for the client. The default value for this property is **modelSingleThread**.

Remarks

The **ThreadModel** property specifies the threading model that is used by the class instance when a connection is established. The default value for this property is **modelSingleThread**, which specifies that only the thread that established the connection should be permitted to invoke methods. It is important to note that this threading model does not limit the application to a single thread of execution. When a session is established using the **Connect** method, that session is attached to the thread that created it. From that point on, until the session is terminated, only the owner may invoke methods in that instance of the class. The ownership of the class instance may be transferred from one thread to another using the **AttachThread** method.

Setting this property to **modelFreeThread** disables certain internal safety checks that are performed by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads. The application must ensure that no two threads will attempt to invoke a blocking method at the same time. In other words, if one thread invokes a method, the application must ensure that another thread will not attempt to invoke any other method at the same time using the same instance of the class.

Changing the value of this property will not affect an active client session. The threading model must be specified prior to invoking the **Connect** method.

See Also

PopClient Class | SocketTools Namespace | AttachThread Method | ThreadingModel Enumeration | ThreadModel Attribute

PopClient.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]

```
Public Property ThrowError As Boolean
```

[C#]

```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

PopClient Class | SocketTools Namespace

PopClient.Timeout Property

Gets and sets a value which specifies a timeout period in seconds.

[Visual Basic]

```
Public Property Timeout As Integer
```

[C#]

```
public int Timeout {get; set;}
```

Property Value

An integer value which specifies a timeout period in seconds.

Remarks

Setting the **Timeout** property specifies the number of seconds until a blocking operation fails and returns an error.

The timeout period is only used when the client is in blocking mode. Although this property can be changed when the client is in non-blocking mode, the value will be ignored until the client is returned to blocking mode.

For most applications it is recommended the timeout period be set between 10 and 20 seconds.

See Also

PopClient Class | SocketTools Namespace

PopClient.Trace Property

Gets and sets a value which indicates if network function logging is enabled.

[Visual Basic]

```
Public Property Trace As Boolean
```

[C#]

```
public bool Trace {get; set;}
```

Property Value

Returns **true** if network function tracing is enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Trace** property is used to enable (or disable) the tracing of network function calls. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the SocketTools networking classes and is enabled or disabled for an entire process. This means that once trace logging is enabled for a given component, all of the function calls made by the process using any of the SocketTools classes will be logged. For example, if you have an application using both the File Transfer Protocol and Post Office Protocol classes, and you set the **Trace** property to **true**, function calls made by both classes will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all classes used by the process.

If trace logging is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the logfile is fairly large. Since logfiles can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

When redistributing your application, make sure that you include the **SocketTools10.TraceLog.dll** module with your installation. If this library is not present, then no trace output will be generated and the value of the **Trace** property will be ignored. Only those function calls made by the SocketTools networking classes will be logged. Calls made directly to the Windows Sockets API, or calls made by other classes, will not be logged.

See Also

PopClient Class | SocketTools Namespace

PopClient.TraceFile Property

Gets and sets a value which specifies the name of the logfile.

[Visual Basic]

```
Public Property TraceFile As String
```

[C#]

```
public string TraceFile {get; set;}
```

Property Value

A string which specifies the name of the file.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named **SocketTools.log** is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFile** property should be set to the same value for each instance. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
 MyApp INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0 MyApp WRN:
connect(46, 192.0.0.1:1234, 16) returned -1 [10035] MyApp ERR: accept(46,
NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the control is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Note that if the specified file cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

PopClient Class | SocketTools Namespace

PopClient.TraceFlags Property

Gets and sets a value which specifies the client function tracing flags.

[Visual Basic]

```
Public Property TraceFlags As TraceOptions
```

[C#]

```
public PopClient.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration which specifies the amount of detail written to the trace logfile.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Because network function tracing is enabled per-process, the trace flags are shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFlags** property should be set to the same value for each instance. Changing the trace flags for any one instance of the class will affect the logging performed for all SocketTools classes used by the application.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being written and the error **errorOperationWouldBlock** occurs, a warning is generated because the application simply needs to attempt to write the data at a later time.

See Also

PopClient Class | SocketTools Namespace

PopClient.UserName Property

Gets and sets the username used to authenticate the client session.

[Visual Basic]

```
Public Property UserName As String
```

[C#]

```
public string UserName {get; set;}
```

Property Value

A string which specifies the username.

Remarks

If a username is not specified when the **Connect** method is called, the value of this property will be used as the default username when establishing a connection with the server.

See Also

PopClient Class | SocketTools Namespace | BearerToken Property | Password Property

PopClient.Version Property

Gets a value which returns the current version of the PopClient class library.

[Visual Basic]

```
Public ReadOnly Property Version As String
```

[C#]

```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the PopClient class library. This value can be used by an application for validation and debugging purposes.

See Also

PopClient Class | SocketTools Namespace

PopClient Methods

The methods of the **PopClient** class are listed below. For a complete list of **PopClient** class members, see the PopClient Members topic.

Public Instance Methods

AttachThread	Attach an instance of the class to the current thread
Cancel	Cancel the current blocking client operation.
ChangePassword	Change the mailbox password for the current user.
CloseMessage	Closes the current message.
Command	Overloaded. Send a custom command to the server.
Connect	Overloaded. Establish a connection with a remote host.
DeleteMessage	Overloaded. Flags a message for deletion from the current mailbox.
Disconnect	Terminate the connection with a remote host.
Dispose	Overloaded. Releases all resources used by PopClient.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetHeader	Overloaded. Returns the value of a header field from the specified message.
GetHeaders	Overloaded. Retrieves the headers for the specified message from the server.
GetMessage	Overloaded. Retrieve a message from the server and return the contents in a byte array.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the PopClient class.
OpenMessage	Overloaded. Open the specified message for reading.
Read	Overloaded. Read data from the server and store it in a byte array.
Reset	Reset the internal state of the object, resetting all properties to their default values.
SendMessage	Overloaded. Submits a message to the mail server for delivery.

StoreMessage	Overloaded. Retrieve a message from the current mailbox and store it in a file on the local system.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
Write	Overloaded. Write one or more bytes of data to the server.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the PopClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

PopClient Class | SocketTools Namespace

PopClient.AttachThread Method

Attach an instance of the class to the current thread

[Visual Basic]
```
Public Function AttachThread() As Boolean
```

[C#]
```
public bool AttachThread();
```

Return Value

A boolean value which specifies if the client could be attached to the current thread. If this method returns **false**, the client could not be attached to the thread and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

When an instance of the class is created it is associated with the current thread that created it. Normally, if another thread attempts to perform an operation using that instance, an error is returned. This is used to ensure that other threads cannot interfere with an operation being performed by the owner thread. In some cases, it may be desirable for one thread in an application to create an instance of the class, establish a connection and then pass that instance to another worker thread. The **AttachThread** method can be used to change the ownership of the class instance to the new worker thread.

This method should be called by the new thread immediately after it has been created, and if the new thread does not release the handle itself, the ownership of the handle should be restored by the original thread. Under no circumstances should **AttachThread** be used to forcibly destroy an instance of a class allocated by another thread while a blocking operation is in progress. To cancel a blocking operation, use the **Cancel** method and then delete the class instance after the blocking function exits and control is returned to the current thread.

See Also

PopClient Class | SocketTools Namespace

PopClient.Cancel Method

Cancel the current blocking client operation.

[Visual Basic]
```
Public Sub Cancel()
```

[C#]
```
public void Cancel();
```

Remarks

When the **Cancel** method is called, the blocking client operation will not immediately fail. An internal flag is set which causes the blocking operation to exit with an error. This means that the application cannot cancel an operation and immediately perform some other blocking function. Instead it must allow the calling stack to unwind, returning back to the blocking operation before making any further function calls.

See Also

PopClient Class | SocketTools Namespace

PopClient.ChangePassword Method

Change the mailbox password for the current user.

[Visual Basic]

```
Public Function ChangePassword( _
   ByVal userName As String, _
   ByVal oldPassword As String, _
   ByVal newPassword As String _
) As Boolean
```

[C#]

```
public bool ChangePassword(
   string userName,
   string oldPassword,
   string newPassword
);
```

Parameters

userName
A string which specifies the username for which the password will be changed.

oldPassword
A string which specifies the current password.

newPassword
A string which specifies the new password.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ChangePassword** method changes the current password that will be used to authenticate the user. Once the password has been changed, the **Password** property will be updated with the new password.

Note that in order to change the user's mailbox password, the server must be running the poppass service on port 106, on the same server. Because passwords are transmitted as clear text (unencrypted), this service is not considered secure and may not be available.

See Also

PopClient Class | SocketTools Namespace

PopClient.CloseMessage Method

Closes the current message.

[Visual Basic]
```
Public Function CloseMessage() As Boolean
```

[C#]
```
public bool CloseMessage();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CloseMessage** method closes the current message. If there is any remaining data left to be read from the message, it will be read and discarded.

See Also

PopClient Class | SocketTools Namespace

PopClient.Command Method

Send a custom command to the server.

Overload List

Send a custom command to the server.

public bool Command(string);

Send a custom command to the server.

public bool Command(string,bool);

Send a custom command to the server.

public bool Command(string,string);

Send a custom command to the server.

public bool Command(string,string,bool);

See Also

PopClient Class | SocketTools Namespace

PopClient.Command Method (String)

Send a custom command to the server.

[Visual Basic]

```
Overloads Public Function Command( _
   ByVal command As String _
) As Boolean
```

[C#]

```
public bool Command(
   string command
);
```

Parameters

command

A string which specifies the command to send. Valid commands vary based on the Internet protocol and the type of server that the client is connected to. Consult the protocol standard and/or the technical reference documentation for the server to determine what commands may be issued by a client application.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Command** method sends a command to the remote host and processes the result code sent back in response to that command. This method can be used to send custom commands to a server to take advantage of features or capabilities that may not be supported internally by the class library.

To determine the specific status code returned by the server, check the value of the **ResultCode** property after the method returns.

See Also

PopClient Class | SocketTools Namespace | PopClient.Command Overload List

PopClient.Command Method (String, Boolean)

Send a custom command to the server.

[Visual Basic]

```
Overloads Public Function Command( _
   ByVal command As String, _
   ByVal isMultiLine As Boolean _
) As Boolean
```

[C#]

```
public bool Command(
   string command,
   bool isMultiLine
);
```

Parameters

command

A string which specifies the command to send. Valid commands vary based on the Internet protocol and the type of server that the client is connected to. Consult the protocol standard and/or the technical reference documentation for the server to determine what commands may be issued by a client application.

isMultiLine

A boolean value which specifies if the command will result in multiple lines of output from the server. For more information about a specific command, consult the standards documentation for the protocol.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Command** method sends a command to the remote host and processes the result code sent back in response to that command. This method can be used to send custom commands to a server to take advantage of features or capabilities that may not be supported internally by the class library.

The *isMultiLine* parameter is used by the method to determine if multiple lines of data will be returned by the server as the result of a command. Unlike a single line response, which consists of a result code and result string, a multi-line response consists of one or more lines of text, terminated by a special end-of-data marker.

To determine the specific status code returned by the server, check the value of the **ResultCode** property after the method returns.

See Also

PopClient Class | SocketTools Namespace | PopClient.Command Overload List

PopClient.Command Method (String, String)

Send a custom command to the server.

[Visual Basic]

```
Overloads Public Function Command( _
   ByVal command As String, _
   ByVal parameters As String _
) As Boolean
```

[C#]

```
public bool Command(
   string command,
   string parameters
);
```

Parameters

command

A string which specifies the command to send. Valid commands vary based on the Internet protocol and the type of server that the client is connected to. Consult the protocol standard and/or the technical reference documentation for the server to determine what commands may be issued by a client application.

parameters

An string which specifies one or more parameters to be sent along with the command. If more than one parameter is required, they must be separated by a single space character. Consult the protocol standard and/or technical reference documentation for the server to determine what parameters should be provided when issuing a specific command.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Command** method sends a command to the remote host and processes the result code sent back in response to that command. This method can be used to send custom commands to a server to take advantage of features or capabilities that may not be supported internally by the class library.

To determine the specific status code returned by the server, check the value of the **ResultCode** property after the method returns.

See Also

PopClient Class | SocketTools Namespace | PopClient.Command Overload List

PopClient.Command Method (String, String, Boolean)

Send a custom command to the server.

[Visual Basic]

```
Overloads Public Function Command( _
   ByVal command As String, _
   ByVal parameters As String, _
   ByVal isMultiLine As Boolean _
) As Boolean
```

[C#]

```
public bool Command(
   string command,
   string parameters,
   bool isMultiLine
);
```

Parameters

command

A string which specifies the command to send. Valid commands vary based on the Internet protocol and the type of server that the client is connected to. Consult the protocol standard and/or the technical reference documentation for the server to determine what commands may be issued by a client application.

parameters

An string which specifies one or more parameters to be sent along with the command. If more than one parameter is required, they must be separated by a single space character. Consult the protocol standard and/or technical reference documentation for the server to determine what parameters should be provided when issuing a specific command.

isMultiLine

A boolean value which specifies if the command will result in multiple lines of output from the server. For more information about a specific command, consult the standards documentation for the protocol.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Command** method sends a command to the remote host and processes the result code sent back in response to that command. This method can be used to send custom commands to a server to take advantage of features or capabilities that may not be supported internally by the class library.

The *isMultiLine* parameter is used by the method to determine if multiple lines of data will be returned by the server as the result of a command. Unlike a single line response, which consists of a result code and result string, a multi-line response consists of one or more lines of text, terminated by a special end-of-data marker.

To determine the specific status code returned by the server, check the value of the **ResultCode** property after the method returns.

See Also

PopClient Class | SocketTools Namespace | PopClient.Command Overload List

PopClient.Connect Method

Establish a connection with a remote host.

Overload List

Establish a connection with a remote host.

public bool Connect();

Establish a connection with a remote host.

public bool Connect(string);

Establish a connection with a remote host.

public bool Connect(string,int);

Establish a connection with a remote host.

public bool Connect(string,int,int);

Establish a connection with a remote host.

public bool Connect(string,int,int,PopOptions);

Establish a connection with a remote host.

public bool Connect(string,int,string,string);

Establish a connection with a remote host.

public bool Connect(string,int,string,string,int);

Establish a connection with a remote host.

public bool Connect(string,int,string,string,int,PopOptions);

Establish a connection with a remote host.

public bool Connect(string,string,string);

See Also

PopClient Class | SocketTools Namespace

PopClient.Connect Method ()

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect() As Boolean
```

[C#]

```
public bool Connect();
```

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection using assigned property values.

The value of the **HostName** or **HostAddress** property will be used to determine the host name or address to connect to.

The value of the **RemotePort** property will be used to determine the port number to connect to.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

See Also

PopClient Class | SocketTools Namespace | PopClient.Connect Overload List

PopClient.Connect Method (String)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

PopClient Class | SocketTools Namespace | PopClient.Connect Overload List

PopClient.Connect Method (String, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

PopClient Class | SocketTools Namespace | PopClient.Connect Overload List

PopClient.Connect Method (String, Int32, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   int timeout
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

PopClient Class | SocketTools Namespace | PopClient.Connect Overload List

PopClient.Connect Method (String, Int32, Int32, PopOptions)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer, _
   ByVal options As PopOptions _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   int timeout,
   PopOptions options
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

options

One or more of the PopOptions enumeration flags.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

PopClient Class | SocketTools Namespace | PopClient.Connect Overload List

PopClient.Connect Method (String, Int32, String, String)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword
);
```

Parameters

hostName
A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort
An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName
A string which specifies the username which will be used to authenticate the client session with the remote host.

userPassword
A string which specifies the password which will be used to authenticate the client session with the remote host. If an OAuth 2.0 authentication method has been specified, this parameter should specify the bearer token.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

PopClient Class | SocketTools Namespace | PopClient.Connect Overload List

PopClient.Connect Method (String, Int32, String, String, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   int timeout
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName

A string which specifies the username which will be used to authenticate the client session with the remote host.

userPassword

A string which specifies the password which will be used to authenticate the client session with the remote host. If an OAuth 2.0 authentication method has been specified, this parameter should specify the bearer token.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

PopClient Class | SocketTools Namespace | PopClient.Connect Overload List

PopClient.Connect Method (String, Int32, String, String, Int32, PopOptions)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal timeout As Integer, _
   ByVal options As PopOptions _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   int timeout,
   PopOptions options
);
```

Parameters

hostName
A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort
An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName
A string which specifies the username which will be used to authenticate the client session with the remote host.

userPassword
A string which specifies the password which will be used to authenticate the client session with the remote host. If an OAuth 2.0 authentication method has been specified, this parameter should specify the bearer token.

timeout
An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

options
One or more of the PopOptions enumeration flags.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

PopClient Class | SocketTools Namespace | PopClient.Connect Overload List

PopClient.Connect Method (String, String, String)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   string userName,
   string userPassword
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

userName

A string which specifies the username which will be used to authenticate the client session with the remote host.

userPassword

A string which specifies the password which will be used to authenticate the client session with the remote host. If an OAuth 2.0 authentication method has been specified, this parameter should specify the bearer token.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

PopClient Class | SocketTools Namespace | PopClient.Connect Overload List

PopClient.DeleteMessage Method

Flags the current message for deletion from the mailbox.

Overload List

Flags the current message for deletion from the mailbox.

public bool DeleteMessage();

Flags a message for deletion from the current mailbox.

public bool DeleteMessage(int);

See Also

PopClient Class | SocketTools Namespace

PopClient.DeleteMessage Method ()

Flags the current message for deletion from the mailbox.

[Visual Basic]

```
Overloads Public Function DeleteMessage() As Boolean
```

[C#]

```
public bool DeleteMessage();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method only flags the message for deletion. The message is not actually deleted until the client disconnects from the server, however the deleted message will no longer be accessible to the client. To prevent deleted messages from actually being removed from the mailbox, call the **Reset** method.

See Also

PopClient Class | SocketTools Namespace | PopClient.DeleteMessage Overload List

PopClient.DeleteMessage Method (Int32)

Flags a message for deletion from the current mailbox.

[Visual Basic]

```
Overloads Public Function DeleteMessage( _
   ByVal messageId As Integer _
) As Boolean
```

[C#]

```
public bool DeleteMessage(
   int messageId
);
```

Parameters

messageId

Number of message to delete from the server. This value must be greater than zero. The first message in the mailbox is message number one.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method only flags the message for deletion. The message is not actually deleted until the client disconnects from the server, however the deleted message will no longer be accessible to the client. To prevent deleted messages from actually being removed from the mailbox, call the **Reset** method.

See Also

PopClient Class | SocketTools Namespace | PopClient.DeleteMessage Overload List

PopClient.Disconnect Method

Terminate the connection with a remote host.

[Visual Basic]

```
Public Sub Disconnect()
```

[C#]

```
public void Disconnect();
```

Remarks

The **Disconnect** method terminates the connection with the remote host and releases the client handle allocated by the class. Note that the socket is not immediately released when the connection is terminated and will enter a wait state for two minutes. After the time wait period has elapsed, the client will be released by the operating system. This is a normal safety mechanism to handle any packets that may arrive after the connection has been closed.

See Also

PopClient Class | SocketTools Namespace

PopClient.Dispose Method

Releases all resources used by PopClient.

Overload List

Releases all resources used by PopClient.

public void Dispose();

Releases the unmanaged resources allocated by the PopClient class and optionally releases the managed resources.

protected virtual void Dispose(bool);

See Also

PopClient Class | SocketTools Namespace

PopClient.Dispose Method ()

Releases all resources used by PopClient.

[Visual Basic]
```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]
```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

PopClient Class | SocketTools Namespace | PopClient.Dispose Overload List

PopClient.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the PopClient class and optionally releases the managed resources.

[Visual Basic]

```
Overridable Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]

```
protected virtual void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **PopClient** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

PopClient Class | SocketTools Namespace | PopClient.Dispose Overload List

PopClient.Finalize Method

Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.

[Visual Basic]

```
Overrides Protected Sub Finalize()
```

[C#]

```
protected override void Finalize();
```

See Also

PopClient Class | SocketTools Namespace

PopClient.GetHeader Method

Returns the value of a header field from the specified message.

Overload List

Returns the value of a header field from the specified message.

public bool GetHeader(int,string,ref string);

Returns the value of a header field from the current message.

public bool GetHeader(string,ref string);

See Also

PopClient Class | SocketTools Namespace

PopClient.GetHeader Method (Int32, String, String)

Returns the value of a header field from the specified message.

[Visual Basic]

```
Overloads Public Function GetHeader( _
   ByVal messageId As Integer, _
   ByVal headerName As String, _
   ByRef headerValue As String _
) As Boolean
```

[C#]

```
public bool GetHeader(
   int messageId,
   string headerName,
   ref string headerValue
);
```

Parameters

messageId
An integer value that specifies the message to retrieve the header value from. This value must be greater than zero. The first message in the mailbox is message number one.

headerName
A string which specifies the message header to retrieve.

headerValue
A string passed by reference which will contain the value of the specified message header if the method is successful.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeader** method returns the value of a header field from the specified message. This allows an application to be able to easily determine the value of a header such as the sender, or the subject of the message. Any header field, including non-standard extensions, may be returned by this method.

See Also

PopClient Class | SocketTools Namespace | PopClient.GetHeader Overload List

PopClient.GetHeader Method (String, String)

Returns the value of a header field from the current message.

[Visual Basic]

```
Overloads Public Function GetHeader( _
   ByVal headerName As String, _
   ByRef headerValue As String _
) As Boolean
```

[C#]

```
public bool GetHeader(
   string headerName,
   ref string headerValue
);
```

Parameters

headerName
A string which specifies the message header to retrieve.

headerValue
A string passed by reference which will contain the value of the specified message header if the method is successful.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeader** method returns the value of a header field from the current message. This allows an application to be able to easily determine the value of a header such as the sender, or the subject of the message. Any header field, including non-standard extensions, may be returned by this method.

The current message number is specified by the value of the **Message** property.

See Also

PopClient Class | SocketTools Namespace | PopClient.GetHeader Overload List

PopClient.GetHeaders Method

Retrieves the headers for the current message from the server.

Overload List

Retrieves the headers for the current message from the server.

public bool GetHeaders(byte[],ref int);

Retrieves the headers for the specified message from the server.

public bool GetHeaders(int,byte[],ref int);

Retrieves the headers for the specified message from the server.

public bool GetHeaders(int,ref string);

Retrieves the headers for the current message from the server.

public bool GetHeaders(ref string);

See Also

PopClient Class | SocketTools Namespace

PopClient.GetHeaders Method (Byte[], Int32)

Retrieves the headers for the current message from the server.

[Visual Basic]

```
Overloads Public Function GetHeaders( _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool GetHeaders(
   byte[] buffer,
   ref int length
);
```

Parameters

buffer

A byte array that will contain the message data when the method returns.

length

An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeaders** method is used to retrieve the message headers from the server and copy it into a local buffer. This method will cause the current thread to block until the article transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Note that the header data will be from the first part of the message, not from any additional sections of a multipart message. In other words, the headers such as From, To, Subject and Date will be returned in the buffer. To retrieve the headers from a specific section of a multipart message, you can use the **GetMessage** method and specify the **ImapSections.sectionHeader** option.

The current message number is specified by the value of the **Message** property.

See Also

PopClient Class | SocketTools Namespace | PopClient.GetHeaders Overload List

PopClient.GetHeaders Method (Int32, Byte[], Int32)

Retrieves the headers for the specified message from the server.

[Visual Basic]

```
Overloads Public Function GetHeaders( _
   ByVal messageId As Integer, _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool GetHeaders(
   int messageId,
   byte[] buffer,
   ref int length
);
```

Parameters

messageId

Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

buffer

A byte array that will contain the message data when the method returns.

length

An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeaders** method is used to retrieve the message headers from the server and copy it into a local buffer. This method will cause the current thread to block until the article transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Note that the header data will be from the first part of the message, not from any additional sections of a multipart message. In other words, the headers such as From, To, Subject and Date will be returned in the buffer. To retrieve the headers from a specific section of a multipart message, you can use the **GetMessage** method and specify the **ImapSections.sectionHeader** option.

See Also

PopClient Class | SocketTools Namespace | PopClient.GetHeaders Overload List

PopClient.GetHeaders Method (Int32, String)

Retrieves the headers for the specified message from the server.

[Visual Basic]

```
Overloads Public Function GetHeaders( _
   ByVal messageId As Integer, _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool GetHeaders(
   int messageId,
   ref string buffer
);
```

Parameters

messageId

Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

buffer

A string passed by reference that will contain the message data when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeaders** method is used to retrieve the message headers from the server and copy it into a local buffer. This method will cause the current thread to block until the article transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Note that the header data will be from the first part of the message, not from any additional sections of a multipart message. In other words, the headers such as From, To, Subject and Date will be returned in the buffer. To retrieve the headers from a specific section of a multipart message, you can use the **GetMessage** method and specify the **ImapSections.sectionHeader** option.

See Also

PopClient Class | SocketTools Namespace | PopClient.GetHeaders Overload List

PopClient.GetHeaders Method (String)

Retrieves the headers for the current message from the server.

[Visual Basic]

```
Overloads Public Function GetHeaders( _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool GetHeaders(
   ref string buffer
);
```

Parameters

buffer
A string passed by reference that will contain the message data when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetHeaders** method is used to retrieve the message headers from the server and copy it into a local buffer. This method will cause the current thread to block until the article transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Note that the header data will be from the first part of the message, not from any additional sections of a multipart message. In other words, the headers such as From, To, Subject and Date will be returned in the buffer. To retrieve the headers from a specific section of a multipart message, you can use the **GetMessage** method and specify the **ImapSections.sectionHeader** option.

The current message number is specified by the value of the **Message** property.

See Also

PopClient Class | SocketTools Namespace | PopClient.GetHeaders Overload List

PopClient.GetMessage Method

Retrieve the current message from the server and return the contents in a byte array.

Overload List

Retrieve the current message from the server and return the contents in a byte array.

public bool GetMessage(byte[],ref int);

Retrieve a message from the server and return the contents in a byte array.

public bool GetMessage(int,byte[],ref int);

Retrieve a message from the server and return the contents in a string.

public bool GetMessage(int,ref string);

Retrieve the current message from the server and return the contents in a string.

public bool GetMessage(ref string);

See Also

PopClient Class | SocketTools Namespace

PopClient.GetMessage Method (Byte[], Int32)

Retrieve the current message from the server and return the contents in a byte array.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool GetMessage(
   byte[] buffer,
   ref int length
);
```

Parameters

buffer

A byte array that the message data will be stored in.

length

An integer value passed by reference which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller. When the method returns, this parameter will be updated with the actual number of bytes copied into the array.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve the current message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

PopClient Class | SocketTools Namespace | PopClient.GetMessage Overload List

PopClient.GetMessage Method (Int32, Byte[], Int32)

Retrieve a message from the server and return the contents in a byte array.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByVal messageId As Integer, _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool GetMessage(
   int messageId,
   byte[] buffer,
   ref int length
);
```

Parameters

messageId
Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

buffer
A byte array that the message data will be stored in.

length
An integer value passed by reference which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller. When the method returns, this parameter will be updated with the actual number of bytes copied into the array.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve a message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

PopClient Class | SocketTools Namespace | PopClient.GetMessage Overload List

PopClient.GetMessage Method (Int32, String)

Retrieve a message from the server and return the contents in a string.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByVal messageId As Integer, _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool GetMessage(
   int messageId,
   ref string buffer
);
```

Parameters

messageId

Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

buffer

A string passed by reference that will contain the message data when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve a message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

PopClient Class | SocketTools Namespace | PopClient.GetMessage Overload List

PopClient.GetMessage Method (String)

Retrieve the current message from the server and return the contents in a string.

[Visual Basic]

```
Overloads Public Function GetMessage( _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool GetMessage(
   ref string buffer
);
```

Parameters

buffer

A string passed by reference that will contain the message data when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetMessage** method is used to retrieve the current message from the server and copy it into a local buffer. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The current message number is specified by the value of the **Message** property.

See Also

PopClient Class | SocketTools Namespace | PopClient.GetMessage Overload List

PopClient.Initialize Method

Initialize an instance of the PopClient class.

Overload List

Initialize an instance of the PopClient class.

public bool Initialize();

Initialize an instance of the PopClient class.

public bool Initialize(string);

See Also

PopClient Class | SocketTools Namespace | Uninitialize Method

PopClient.Initialize Method ()

Initialize an instance of the PopClient class.

[Visual Basic]
```
Overloads Public Function Initialize() As Boolean
```

[C#]
```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the PopClient class, loading the networking library and allocating resources for the current thread. Typically it is not necessary to explicitly call this method because the instance of the class is initialized by the class constructor. However, if the **Uninitialize** method is called, the class must be re-initialized before any other methods are called.

See Also

PopClient Class | SocketTools Namespace | PopClient.Initialize Overload List | Uninitialize Method

PopClient.Initialize Method (String)

Initialize an instance of the PopClient class.

[Visual Basic]
```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]
```
public bool Initialize(
   string licenseKey
);
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the PopClient class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the PopClient class.

Example

The following example shows how to use the Initialize method to initialize an instance of the class. This example assumes that the license key string has been defined in code.

```
SocketTools.PopClient popClient = new SocketTools.PopClient();

if (popClient.Initialize(strLicenseKey) == false)
{
    MessageBox.Show(popClient.LastErrorString, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    return;
}
```

```
Dim popClient As New SocketTools.PopClient

If popClient.Initialize(strLicenseKey) = False Then
    MsgBox(popClient.LastErrorString, vbIconExclamation)
    Exit Sub
End If
```

See Also

PopClient Class | SocketTools Namespace | PopClient.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

PopClient.OpenMessage Method

Open the current message for reading.

Overload List

Open the current message for reading.

public bool OpenMessage();

Open the specified message for reading.

public bool OpenMessage(int);

See Also

PopClient Class | SocketTools Namespace

PopClient.OpenMessage Method ()

Open the current message for reading.

[Visual Basic]

```
Overloads Public Function OpenMessage() As Boolean
```

[C#]

```
public bool OpenMessage();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **OpenMessage** method opens the current message in the mailbox. The client can then read the contents of the message using the **Read** method, and once all of the data has been read, the message should be closed by calling the **CloseMessage** method.

The current message number is specified by the value of the **Message** property.

See Also

PopClient Class | SocketTools Namespace | PopClient.OpenMessage Overload List

PopClient.OpenMessage Method (Int32)

Open the specified message for reading.

[Visual Basic]

```
Overloads Public Function OpenMessage( _
   ByVal messageId As Integer _
) As Boolean
```

[C#]

```
public bool OpenMessage(
   int messageId
);
```

Parameters

messageId

Number of article to retrieve from the server. This value must be greater than zero. The first message in the mailbox is message number one.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **OpenMessage** method opens a message in the current mailbox. The client can then read the contents of the message using the **Read** method, and once all of the data has been read, the message should be closed by calling the **CloseMessage** method.

See Also

PopClient Class | SocketTools Namespace | PopClient.OpenMessage Overload List

PopClient.Read Method

Read data from the server and store it in a byte array.

Overload List

Read data from the server and store it in a byte array.

public int Read(byte[]);

Read data from the server and store it in a byte array.

public int Read(byte[],int);

Read data from the server and store it in a string.

public int Read(ref string);

Read data from the server and store it in a string.

public int Read(ref string,int);

See Also

PopClient Class | SocketTools Namespace

PopClient.Read Method (Byte[])

Read data from the server and store it in a byte array.

```
[Visual Basic]
Overloads Public Function Read( _
   ByVal buffer As Byte() _
) As Integer
```

```
[C#]
public int Read(
   byte[] buffer
);
```

Parameters

buffer

A byte array that the data will be stored in.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the size of the byte array passed to the method. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

See Also

PopClient Class | SocketTools Namespace | PopClient.Read Overload List

PopClient.Read Method (Byte[], Int32)

Read data from the server and store it in a byte array.

[Visual Basic]

```
Overloads Public Function Read( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Read(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array that the data will be stored in.

length

An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the number of bytes specified. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

See Also

PopClient Class | SocketTools Namespace | PopClient.Read Overload List

PopClient.Read Method (String)

Read data from the server and store it in a string.

[Visual Basic]

```
Overloads Public Function Read( _
   ByRef buffer As String _
) As Integer
```

[C#]

```
public int Read(
   ref string buffer
);
```

Parameters

buffer

A string that will contain the data read from the client.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to a maximum of 4096 bytes. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

See Also

PopClient Class | SocketTools Namespace | PopClient.Read Overload List

PopClient.Read Method (String, Int32)

Read data from the server and store it in a string.

```
[Visual Basic]
Overloads Public Function Read( _
   ByRef buffer As String, _
   ByVal length As Integer _
) As Integer
```

```
[C#]
public int Read(
   ref string buffer,
   int length
);
```

Parameters

buffer

A string that will contain the data read from the client.

length

An integer value which specifies the maximum number of bytes of data to read.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the number of bytes specified. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

See Also

PopClient Class | SocketTools Namespace | PopClient.Read Overload List

PopClient.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. If a socket has been allocated, it will be released and any active connections will be terminated. All properties will be reset to their default values.

See Also

PopClient Class | SocketTools Namespace

PopClient.SendMessage Method

Submits a message to the mail server for delivery.

Overload List

Submits a message to the mail server for delivery.

public bool SendMessage(byte[],int);

Submits a message to the mail server for delivery.

public bool SendMessage(string);

See Also

PopClient Class | SocketTools Namespace

PopClient.SendMessage Method (Byte[], Int32)

Submits a message to the mail server for delivery.

[Visual Basic]

```
Overloads Public Function SendMessage( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool SendMessage(
   byte[] buffer,
   int length
);
```

Parameters

buffer
A byte array which contains the message to be submitted for delivery.

length
An integer value which specifies the maximum number of bytes of data to send. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SendMessage** method submits a message to the mail server for delivery. The message format must comply with the RFC 822 standard, with the header and body separated by a blank line, and each line terminated with carriage-return/linefeed characters.

This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Note that this method requires that the server support the XTND XMIT command. Although using this method to send mail has the advantage that the sender is authenticated (because the user must first login to the server), it is not widely supported. For general purpose mail delivery service, it is recommended that an application use the Simple Mail Transfer Protocol (SMTP).

See Also

PopClient Class | SocketTools Namespace | PopClient.SendMessage Overload List

PopClient.SendMessage Method (String)

Submits a message to the mail server for delivery.

[Visual Basic]

```
Overloads Public Function SendMessage( _
   ByVal buffer As String _
) As Boolean
```

[C#]

```
public bool SendMessage(
   string buffer
);
```

Parameters

buffer

A string which contains the message to be submitted for delivery.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SendMessage** method submits a message to the mail server for delivery. The message format must comply with the RFC 822 standard, with the header and body separated by a blank line, and each line terminated with carriage-return/linefeed characters.

This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Note that this method requires that the server support the XTND XMIT command. Although using this method to send mail has the advantage that the sender is authenticated (because the user must first login to the server), it is not widely supported. For general purpose mail delivery service, it is recommended that an application use the Simple Mail Transfer Protocol (SMTP).

See Also

PopClient Class | SocketTools Namespace | PopClient.SendMessage Overload List

PopClient.StoreMessage Method

Retrieve a message from the current mailbox and store it in a file on the local system.

Overload List

Retrieve a message from the current mailbox and store it in a file on the local system.

public bool StoreMessage(int,string);

Retrieve the current message and store it in a file on the local system.

public bool StoreMessage(string);

See Also

PopClient Class | SocketTools Namespace

PopClient.StoreMessage Method (Int32, String)

Retrieve a message from the current mailbox and store it in a file on the local system.

[Visual Basic]

```
Overloads Public Function StoreMessage( _
   ByVal messageId As Integer, _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool StoreMessage(
   int messageId,
   string fileName
);
```

Parameters

messageId

Number of message to retrieve. This value must be greater than zero. The first message in the mailbox is message number one.

fileName

A string which specifies the file that the message will be stored in. If the file does not exist, it will be created. If the file does exist, it will be overwritten with the contents of the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **StoreMessage** method retrieves a message from the server and stores it in a file on the local system. The contents of the message is stored as a text file, using the specified file name. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

PopClient Class | SocketTools Namespace | PopClient.StoreMessage Overload List

PopClient.StoreMessage Method (String)

Retrieve the current message and store it in a file on the local system.

[Visual Basic]

```
Overloads Public Function StoreMessage( _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool StoreMessage(
   string fileName
);
```

Parameters

fileName

A string which specifies the file that the message will be stored in. If the file does not exist, it will be created. If the file does exist, it will be overwritten with the contents of the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **StoreMessage** method retrieves the current message from the server and stores it in a file on the local system. The contents of the message is stored as a text file, using the specified file name. This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The current message number is specified by the value of the **Message** property.

See Also

PopClient Class | SocketTools Namespace | PopClient.StoreMessage Overload List

PopClient.Uninitialize Method

Uninitialize the class library and release any resources allocated for the current thread.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method terminates any active connection, releases resources allocated for the current thread and unloads the networking library. After this method has been called, no further client operations may be performed until the class instance has been re-initialized.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the class is no longer needed.

See Also

PopClient Class | SocketTools Namespace | Initialize Method

PopClient.Write Method

Write one or more bytes of data to the server.

Overload List

Write one or more bytes of data to the server.

public int Write(byte[]);

Write one or more bytes of data to the server.

public int Write(byte[],int);

Write a string of characters to the server.

public int Write(string);

Write a string of characters to the server.

public int Write(string,int);

See Also

PopClient Class | SocketTools Namespace

PopClient.Write Method (Byte[])

Write one or more bytes of data to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Write(
   byte[] buffer
);
```

Parameters

buffer

A byte array that contains the data to be written to the server.

Return Value

An integer value which specifies the number of bytes actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

PopClient Class | SocketTools Namespace | PopClient.Write Overload List

PopClient.Write Method (Byte[], Int32)

Write one or more bytes of data to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Write(
   byte[] buffer,
   int length
);
```

Parameters

buffer
A byte array that contains the data to be written to the server.

length
An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

PopClient Class | SocketTools Namespace | PopClient.Write Overload List

PopClient.Write Method (String)

Write a string of characters to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String _
) As Integer
```

[C#]

```
public int Write(
   string buffer
);
```

Parameters

buffer

A string which contains the data to be written to the server.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

PopClient Class | SocketTools Namespace | PopClient.Write Overload List

PopClient.Write Method (String, Int32)

Write a string of characters to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Write(
   string buffer,
   int length
);
```

Parameters

buffer

A string which contains the data to be written to the server.

length

An integer value which specifies the maximum number of characters to write. This value cannot be larger than the length of the string specified by the caller.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

PopClient Class | SocketTools Namespace | PopClient.Write Overload List

PopClient Events

The events of the **PopClient** class are listed below. For a complete list of **PopClient** class members, see the PopClient Members topic.

Public Instance Events

⚡ OnCancel	Occurs when a blocking client operation is canceled.
⚡ OnConnect	Occurs when a connection is established with the remote host.
⚡ OnDisconnect	Occurs when the remote host disconnects from the local system.
⚡ OnError	Occurs when an client operation fails.
⚡ OnProgress	Occurs as a data stream is being read or written to the client.
⚡ OnRead	Occurs when data is available to be read from the client.
⚡ OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
⚡ OnWrite	Occurs when data can be written to the client.

See Also

PopClient Class | SocketTools Namespace

PopClient.OnCancel Event

Occurs when a blocking client operation is canceled.

[Visual Basic]

```
Public Event OnCancel As EventHandler
```

[C#]

```
public event EventHandler OnCancel;
```

Remarks

The **OnCancel** event is generated when a blocking client operation, such as sending or receiving data, is canceled with the **Cancel** method. To assist in determining which operation was canceled, check the value of the **Status** property.

See Also

PopClient Class | SocketTools Namespace

PopClient.OnConnect Event

Occurs when a connection is established with the remote host.

[Visual Basic]

```
Public Event OnConnect As EventHandler
```

[C#]

```
public event EventHandler OnConnect;
```

Remarks

The **OnConnect** event occurs when a connection is made with a remote host as a result of a **Connect** method call. When the **Connect** method is called and the **Blocking** property is set to **false**, a socket is created but the connection is not actually established until after this event occurs. Between the time connection process is started and this event fires, no operation may be performed on the client other than calling the **Disconnect** method.

This event is only generated if the client is in non-blocking mode.

See Also

PopClient Class | SocketTools Namespace

PopClient.OnDisconnect Event

Occurs when the remote host disconnects from the local system.

[Visual Basic]

```
Public Event OnDisconnect As EventHandler
```

[C#]

```
public event EventHandler OnDisconnect;
```

Remarks

The **OnDisconnect** event occurs when the remote host closes its connection, terminating the client session with the application. Because there may still be data in the client receive buffers, you should continue to read data from the client until the **Read** method returns a value of 0. Once all of the data has been read, you should call the **Disconnect** method to close the local socket and release the resources allocated for the client.

This event is only generated if the client is in non-blocking mode.

See Also

PopClient Class | SocketTools Namespace

PopClient.OnError Event

Occurs when an client operation fails.

[Visual Basic]
```
Public Event OnError As OnErrorEventHandler
```

[C#]
```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type PopClient.ErrorEventArgs containing data related to this event. The following **PopClient.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a client operation fails.

See Also

PopClient Class | SocketTools Namespace

PopClient.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see PopClient.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.PopClient.ErrorEventArgs

[Visual Basic]

```
Public Class PopClient.ErrorEventArgs
    Inherits EventArgs
```

[C#]

```
public class PopClient.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.PopClient (in SocketTools.PopClient.dll)

See Also

PopClient.ErrorEventArgs Members | SocketTools Namespace

PopClient.ErrorEventArgs Members

PopClient.ErrorEventArgs overview

Public Instance Constructors

PopClient.ErrorEventArgs Constructor	Initializes a new instance of the PopClient.ErrorEventArgs class.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

PopClient.ErrorEventArgs Class | SocketTools Namespace

PopClient.ErrorEventArgs Constructor

Initializes a new instance of the PopClient.ErrorEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public PopClient.ErrorEventArgs();
```

See Also

PopClient.ErrorEventArgs Class | SocketTools Namespace

PopClient.ErrorEventArgs Properties

The properties of the **PopClient.ErrorEventArgs** class are listed below. For a complete list of **PopClient.ErrorEventArgs** class members, see the PopClient.ErrorEventArgs Members topic.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

PopClient.ErrorEventArgs Class | SocketTools Namespace

PopClient.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

PopClient.ErrorEventArgs Class | SocketTools Namespace | Error Property

PopClient.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public PopClient.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

PopClient.ErrorEventArgs Class | SocketTools Namespace | Description Property

PopClient.OnProgress Event

Occurs as a data stream is being read or written to the client.

[Visual Basic]
```
Public Event OnProgress As OnProgressEventHandler
```

[C#]
```
public event OnProgressEventHandler OnProgress;
```

Event Data

The event handler receives an argument of type PopClient.ProgressEventArgs containing data related to this event. The following **PopClient.ProgressEventArgs** properties provide information specific to this event.

Property	Description
BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
BytesTotal	Gets a value which specifies the total number of bytes in the data stream.
Message	Gets the message number.
Percent	Gets a value which specifies the percentage of data that has been read or written.

Remarks

The **OnProgress** event occurs as a data stream is being read or written to the client. If large amounts of data are being read or written, this event can be used to update a progress bar or other user-interface component to provide the user with some visual feedback on the progress of the operation.

See Also

PopClient Class | SocketTools Namespace

PopClient.ProgressEventArgs Class

Provides data for the OnProgress event.

For a list of all members of this type, see PopClient.ProgressEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.PopClient.ProgressEventArgs

[Visual Basic]
```
Public Class PopClient.ProgressEventArgs
    Inherits EventArgs
```

[C#]
```
public class PopClient.ProgressEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ProgressEventArgs specifies the number of bytes copied from the data stream, the total number of bytes in the data stream and a completion percentage.

The OnProgress event occurs as a data stream is being read or written to the client.

Requirements

Namespace: SocketTools

Assembly: SocketTools.PopClient (in SocketTools.PopClient.dll)

See Also

PopClient.ProgressEventArgs Members | SocketTools Namespace

PopClient.ProgressEventArgs Members

PopClient.ProgressEventArgs overview

Public Instance Constructors

PopClient.ProgressEventArgs Constructor	Initializes a new instance of the PopClient.ProgressEventArgs class.

Public Instance Properties

BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
BytesTotal	Gets a value which specifies the total number of bytes in the data stream.
Message	Gets the message number.
Percent	Gets a value which specifies the percentage of data that has been read or written.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

PopClient.ProgressEventArgs Class | SocketTools Namespace

PopClient.ProgressEventArgs Constructor

Initializes a new instance of the PopClient.ProgressEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public PopClient.ProgressEventArgs();
```

See Also

PopClient.ProgressEventArgs Class | SocketTools Namespace

PopClient.ProgressEventArgs Properties

The properties of the **PopClient.ProgressEventArgs** class are listed below. For a complete list of **PopClient.ProgressEventArgs** class members, see the PopClient.ProgressEventArgs Members topic.

Public Instance Properties

BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
BytesTotal	Gets a value which specifies the total number of bytes in the data stream.
Message	Gets the message number.
Percent	Gets a value which specifies the percentage of data that has been read or written.

See Also

PopClient.ProgressEventArgs Class | SocketTools Namespace

PopClient.ProgressEventArgs.BytesCopied Property

Gets a value which specifies the number of bytes of data that has been read or written.

[Visual Basic]
```
Public ReadOnly Property BytesCopied As Integer
```

[C#]
```
public int BytesCopied {get;}
```

Property Value

An integer value which specifies the number of bytes of data.

Remarks

The **BytesCopied** property specifies the number of bytes that have been read from the client and stored in the local stream buffer, or written from the stream buffer to the client.

See Also

PopClient.ProgressEventArgs Class | SocketTools Namespace | BytesTotal Property | Percent Property

PopClient.ProgressEventArgs.BytesTotal Property

Gets a value which specifies the total number of bytes in the data stream.

[Visual Basic]

```
Public ReadOnly Property BytesTotal As Integer
```

[C#]

```
public int BytesTotal {get;}
```

Property Value

An integer value which specifies the number of bytes of data.

Remarks

The **BytesTotal** property specifies the total amount of data being read from the client and stored in the data stream, or written from the data stream to the client. If the amount of data was unknown or unspecified at the time the method call was made, then this value will always be the same as the **BytesCopied** property.

See Also

PopClient.ProgressEventArgs Class | SocketTools Namespace | BytesCopied Property | Percent Property

PopClient.ProgressEventArgs.Message Property

Gets the message number.

[Visual Basic]
```
Public ReadOnly Property Message As Integer
```

[C#]
```
public int Message {get;}
```

Property Value

An integer value which specifies the message number.

Remarks

The **Message** property specifies the message number for the current message that is being downloaded from the mail server to the local host. If the **OnProgress** event occurs while message data is being uploaded to the server, this property will return a value of zero.

See Also

PopClient.ProgressEventArgs Class | SocketTools Namespace

PopClient.ProgressEventArgs.Percent Property

Gets a value which specifies the percentage of data that has been read or written.

[Visual Basic]

```
Public ReadOnly Property Percent As Integer
```

[C#]

```
public int Percent {get;}
```

Property Value

An integer value which specifies a percentage.

Remarks

The **Percent** property specifies the percentage of data that has been transmitted, expressed as an integer value between 0 and 100, inclusive. If the maximum size of the data stream was not specified by the caller, this value will always be 100.

See Also

PopClient.ProgressEventArgs Class | SocketTools Namespace | BytesCopied Property | BytesTotal Property

PopClient.OnRead Event

Occurs when data is available to be read from the client.

[Visual Basic]
```
Public Event OnRead As EventHandler
```

[C#]
```
public event EventHandler OnRead;
```

Remarks

The **OnRead** event occurs when data is available to be read from the client. This event is level-triggered, which means that once this event fires, it will not occur again until some data has been read from the client. This design prevents an application from being flooded with event notifications. It is recommended that your application read all of the available data from the server and store it in a local buffer for processing. See the example below.

This event is only generated if the client is in non-blocking mode.

Example

```
Private Sub Socket_OnRead(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Socket.OnRead
    Dim strBuffer As String
    Dim nRead As Integer

    Do
        ' Read up to m_nBufferSize bytes of data from the server
        nRead = Socket.Read(strBuffer, m_nBufferSize)

        If nRead > 0 Then
            ' Append the data to an internal buffer for processing
            m_dataBuffer = m_dataBuffer + strBuffer
        End If
    Loop Until nRead < 1

    ProcessData()
End Sub
```

See Also

PopClient Class | SocketTools Namespace

PopClient.OnTimeout Event

Occurs when a blocking operation fails to complete before the timeout period elapses.

[Visual Basic]

```
Public Event OnTimeout As EventHandler
```

[C#]

```
public event EventHandler OnTimeout;
```

Remarks

The **OnTimeout** event occurs when a blocking operation, such as sending or receiving data on the client, fails to complete before the specified timeout period elapses. The timeout period for a blocking operation can be adjusted by setting the **Timeout** property.

This event is only generated if the client is in blocking mode.

See Also

PopClient Class | SocketTools Namespace

PopClient.OnWrite Event

Occurs when data can be written to the client.

[Visual Basic]

```
Public Event OnWrite As EventHandler
```

[C#]

```
public event EventHandler OnWrite;
```

Remarks

The **OnWrite** event occurs when the application can write data to the client. This event will typically occur when a connection is first established with the remote host, and after the **Write** method has failed because there was insufficient memory available in the client send buffers. In the second case, when some of the buffered data has been successfully sent to the remote host and there is space available in the send buffers, this event is used to signal the application that it may attempt to send more data.

This event is only generated if the client is in non-blocking mode.

See Also

PopClient Class | SocketTools Namespace

PopClient.ErrorCode Enumeration

Specifies the error codes returned by the PopClient class.

[Visual Basic]
```
Public Enum PopClient.ErrorCode
```

[C#]
```
public enum PopClient.ErrorCode
```

Remarks

The PopClient class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.
errorProxyAuthenticationRequired	Proxy authentication required.

errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.
errorUnsupportedMediaType	Request specified an unsupported media type.

errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.
errorNoMoreMailboxes	No more mailboxes exist on this server.

errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.
errorInvalidThreadHandle	The specified thread handle is no longer valid.

errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been defined.

errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or unrecognized.

errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.

Requirements

Namespace: SocketTools

Assembly: SocketTools.PopClient (in SocketTools.PopClient.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

PopClient.PopAuthentication Enumeration

Specifies the authentication methods supported by the PopClient class.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum PopClient.PopAuthentication
```

[C#]

```
[Flags]
public enum PopClient.PopAuthentication
```

Members

Member Name	Description	Value
authDefault	The default authentication scheme which sends the username and password as cleartext to the server. Because the user credentials are not encrypted, this method should only be used over a secure connection. This is the same as specifying **authPassword** as the authentication method.	0
authPassword	The username and password is sent to the server using the USER and PASS commands. This authentication method is supported by most servers and is the default authentication type. The credentials are not encrypted and this method should only be used over secure connections.	0
authApop	The APOP authentication method which uses an MD5 digest of the password. This method has been deprecated is not supported by all servers. It should only be used if required by legacy mail servers which do not support the SASL authentication methods.	1
authLogin	This authentication type will use the LOGIN method to authenticate the client session. This encodes the username and password in a specific format, but the credentials are not encrypted. It should be used over a secure connection. The server must support the Simple Authentication and Security Layer (SASL) mechanism as defined in RFC 4422.	3
authPlain	This authentication type will use the	4

	PLAIN method to authenticate the client session. This encodes the username and password in a specific format, but the credentials are not encrypted. It should be used over a secure connection. The server must support the PLAIN Simple Authentication and Security Layer (SASL) mechanism as defined in RFC 4616.	
authXOAuth2	This authentication type will use the XOAUTH2 method to authenticate the client session. This authentication method does not require the user password, instead the BearerToken property must specify the OAuth 2.0 bearer token issued by the service provider.	6
authBearer	This authentication type will use the OAUTHBEARER method to authenticate the client session as defined in RFC 7628. This authentication method does not require the user password, instead the BearerToken property must specify the OAuth 2.0 bearer token issued by the service provider.	7

Requirements

Namespace: SocketTools

Assembly: SocketTools.PopClient (in SocketTools.PopClient.dll)

See Also

SocketTools Namespace

PopClient.PopOptions Enumeration

Specifies the options that the PopClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum PopClient.PopOptions
```

[C#]

```
[Flags]
public enum PopClient.PopOptions
```

Remarks

The PopClient class uses the **PopOptions** enumeration to specify one or more options to be used when establishing a connection with a remote host. Multiple options may be specified if necessary.

Members

Member Name	Description	Value
optionNone	No option specified.	0
optionDefault	The default connection option. This is the same as specifying **optionNone**.	0
optionLineBreak	Message data that is received from the server is read as individual lines of text terminated by a carriage return and linefeed control sequence. This option can be useful for applications that need to use the lower level network I/O functions and must process the message text on a line-by-line basis. This option is not recommended for most applications because it can have a negative impact on performance when retrieving large messages from the server.	1
optionTunnel	This option specifies that a tunneled TCP connection and/or port-forwarding is being used to establish the connection to the server. This changes the behavior of the client with regards to internal checks of the destination IP address and remote port number, default capability selection and how the connection is established. This option also forces all connections to be outbound and enables the firewall compatibility features in the client.	1024
optionTrustedSite	This option specifies the server is	2048

	trusted. The server certificate will not be validated and the connection will always be permitted. This option only affects connections using either the SSL or TLS protocols.	
optionSecure	This option specifies the client should attempt to establish a secure connection with the server. The server must support secure connections using either the SSL or TLS protocol.	4096
optionExplicitSSL	This option specifies the client should attempt to establish a secure explicit SSL session. The initial connection to the server is not encrypted, and the client will attempt to negotiate a secure connection by sending the STLS command to the server. Some servers may require this option when connecting to the server on ports other than the default secure port of 995.	4096
optionImplicitSSL	This option specifies the client should attempt to establish a secure implicit SSL session. The SSL handshake is initiated immediately after the connection to the server has been established.	8192
optionSecureFallback	This option specifies the client should permit the use of less secure cipher suites for compatibility with legacy servers. If this option is specified, the client will permit connections using TLS 1.0 and cipher suites that use RC4, MD5 and SHA1.	32768
optionPreferIPv6	This option specifies the client should prefer the use of IPv6 if the server hostname can be resolved to both an IPv6 and IPv4 address. This option is ignored if the local system does not have IPv6 enabled, or when the hostname can only be resolved to an IPv4 address. If the server hostname can only be resolved to an IPv6 address, the client will attempt to establish a connection using IPv6 regardless if this option has been specified.	262144
optionFreeThread	This option specifies that class methods may be called from any thread, and not only the thread that established the connection. Using this option disables	524288

	certain internal safety checks that are made by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads.	

Requirements

Namespace: SocketTools

Assembly: SocketTools.PopClient (in SocketTools.PopClient.dll)

See Also

SocketTools Namespace

PopClient.PopStatus Enumeration

Specifies the status values that may be returned by the PopClient class.

[Visual Basic]
```
Public Enum PopClient.PopStatus
```

[C#]
```
public enum PopClient.PopStatus
```

Remarks

The PopClient class uses the **PopStatus** enumeration to identify the current status of the client.

Members

Member Name	Description
statusUnused	A client session has not been created. Attempts to perform any network operations, such as sending or receiving data, will generate an error.
statusIdle	A client session has been created, but is not currently in use. A blocking socket operation can be executed at this point.
statusConnect	The client is in the process of establishing a connection with a remote host.
statusRead	The client is in the process of receiving data from a remote host.
statusWrite	The client is in the process of sending data to a remote host.
statusDisconnect	The client session is being closed and subsequent attempts to access the client will result in an error.

Requirements

Namespace: SocketTools

Assembly: SocketTools.PopClient (in SocketTools.PopClient.dll)

See Also

SocketTools Namespace

PopClient.SecureCipherAlgorithm Enumeration

Specifies the encryption algorithms that the PopClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum PopClient.SecureCipherAlgorithm
```

```
[C#]
[Flags]
public enum PopClient.SecureCipherAlgorithm
```

Remarks

The PopClient class uses the **SecureCipherAlgorithm** enumeration to identify which encryption algorithm was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
cipherNone	No cipher has been selected. A secure connection has not been established with the remote host.	0
cipherRC2	The RC2 block cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.	1
cipherRC4	The RC4 stream cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.	2
cipherRC5	The RC5 block cipher was selected. This is a variable key length cipher which supports keys up to 2040 bits, in 8-bit increments.	4
cipherDES	The DES (Data Encryption Standard) block cipher was selected. This is a fixed key length cipher using 56-bit keys.	8
cipherDES3	The Triple DES block cipher was selected. This cipher encrypts the data three times using different keys, effectively using a 168-bit key length.	16
cipherDESX	A variant of the DES block cipher which XORs an extra 64-bits of the key before and after the plaintext has been encrypted, increasing the key size to 184 bits.	32
cipherAES	The Advanced Encryption Standard	64

	cipher (also known as the Rijndael cipher) is a fixed block size cipher which use a key size of 128, 192 or 256 bits. This cipher is supported on Windows XP SP3 SP3 and later versions of the operating system.	
cipherSkipjack	The Skipjack block cipher was selected. This is a fixed key length cipher, using 80-bit keys.	128

Requirements

Namespace: SocketTools

Assembly: SocketTools.PopClient (in SocketTools.PopClient.dll)

See Also

SocketTools Namespace

PopClient.SecureHashAlgorithm Enumeration

Specifies the hash algorithms that the PopClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum PopClient.SecureHashAlgorithm
```

[C#]

```
[Flags]
public enum PopClient.SecureHashAlgorithm
```

Remarks

The PopClient class uses the **SecureHashAlgorithm** enumeration to identify the message digest (hash) algorithm that was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
hashNone	No hash algorithm has been selected. This is not a secure connection with the server.	0
hashMD5	The MD5 algorithm was selected. This algorithm produces a 128-bit message digest. This algorithm is no longer considered to be cryptographically secure.	1
hashSHA	The SHA-1 algorithm was selected. This algorithm produces a 160-bit message digest. This algorithm is no longer considered to be cryptographically secure.	2
hashSHA256	The SHA-256 algorithm was selected. This algorithm produces a 256-bit message digest.	4
hashSHA384	The SHA-384 algorithm was selected. This algorithm produces a 384-bit message digest.	8
hashSHA512	The SHA-512 algorithm was selected. This algorithm produces a 512-bit message digest.	16

Requirements

Namespace: SocketTools

Assembly: SocketTools.PopClient (in SocketTools.PopClient.dll)

See Also

SocketTools Namespace

PopClient.SecureKeyAlgorithm Enumeration

Specifies the key exchange algorithms that the PopClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum PopClient.SecureKeyAlgorithm
```

[C#]

```
[Flags]
public enum PopClient.SecureKeyAlgorithm
```

Remarks

The PopClient class uses the **SecureKeyAlgorithm** enumeration to identify the key exchange algorithm that was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
keyExchangeNone	No key exchange algorithm has been selected. This is not a secure connection with the server.	0
keyExchangeRSA	The RSA public key exchange algorithm has been selected.	1
keyExchangeKEA	The KEA public key exchange algorithm has been selected. This is an improved version of the Diffie-Hellman public key algorithm.	2
keyExchangeDH	The Diffie-Hellman public key exchange algorithm has been selected.	4
keyExchangeECDH	The Elliptic Curve Diffie-Hellman key exchange algorithm was selected. This is a variant of the Diffie-Hellman algorithm which uses elliptic curve cryptography. This key exchange algorithm is only supported on Windows XP SP3 SP3 and later versions of the operating system.	8

Requirements

Namespace: SocketTools

Assembly: SocketTools.PopClient (in SocketTools.PopClient.dll)

See Also

SocketTools Namespace

PopClient.SecurityCertificate Enumeration

Specifies the security certificate status values that may be returned by the PopClient class.

[Visual Basic]
```
Public Enum PopClient.SecurityCertificate
```

[C#]
```
public enum PopClient.SecurityCertificate
```

Remarks

The PopClient class uses the **SecurityCertificate** enumeration to identify the current status of the certificate that was provided by the remote host when a secure connection was established.

Members

Member Name	Description
certificateNone	No certificate information is available. A secure connection was not established with the server.
certificateValid	The certificate is valid.
certificateNoMatch	The certificate is valid, however the domain name specified in the certificate does not match the domain name of the remote host. The application can examine the **CertificateSubject** property to determine the site the certificate was issued to.
certificateExpired	The certificate has expired and is no longer valid. The application can examine the **CertificateExpires** property to determine when the certificate expired.
certificateRevoked	The certificate has been revoked and is no longer valid. It is recommended that the application immediately terminate the connection if this status is returned.
certificateUntrusted	The certificate has not been issued by a trusted authority, or the certificate is not trusted on the local host. It is recommended that the application immediately terminate the connection if this status is returned.
certificateInvalid	The certificate is invalid. This typically indicates that the internal structure of the certificate is damaged. It is recommended that the application immediately terminate the connection if this status is returned.

Requirements

Namespace: SocketTools

Assembly: SocketTools.PopClient (in SocketTools.PopClient.dll)

See Also

SocketTools Namespace

PopClient.SecurityProtocols Enumeration

Specifies the security protocols that the PopClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum PopClient.SecurityProtocols
```

[C#]

```
[Flags]
public enum PopClient.SecurityProtocols
```

Remarks

The PopClient class uses the **SecurityProtocols** enumeration to specify one or more security protocols to be used when establishing a connection with a remote host. Multiple protocols may be specified if necessary and the actual protocol used will be negotiated with the remote host. It is recommended that most applications use **protocolDefault** when creating a secure connection.

Members

Member Name	Description	Value
protocolNone	No security protocol will be used, a secure connection will not be established.	0
protocolSSL2	The SSL 2.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections.	1
protocolSSL3	The SSL 3.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections. In most cases, this protocol is only selected if TLS is not supported by the server.	2
protocolTLS10	The TLS 1.0 protocol has been selected. This version of the protocol is commonly used by older servers and is the only version of TLS supported on Windows platforms prior to Windows 7 SP1 and Windows Server 2008 R2.	4
protocolTLS11	The TLS 1.1 protocol has been selected. This version of TLS is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of the	8

	operating system.	
protocolTLS12	The TLS 1.2 protocol has been selected. This is the default version of the protocol and is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of Windows. It is recommended that you use this version of TLS.	16
protocolSSL	Any version of the Secure Sockets Layer (SSL) protocol should be used. The actual protocol version used will be negotiated with the remote host.	3
protocolTLS	Any version of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with with preference for TLS 1.2.	28
protocolTLS1	Version 1.0, 1.1 or 1.2 of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with preference for TLS 1.2.	28
protocolDefault	The default selection of security protocols will be used when establishing a connection. The TLS 1.2, 1.1 and 1.0 protocols will be negotiated with the host, in that order of preference. This option will always request the latest version of the preferred security protocols and is the recommended value.	16
protocolUnknown	An unknown or unsupported security protocol has been specified. This value indicates an error condition.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.PopClient (in SocketTools.PopClient.dll)

See Also

SocketTools Namespace

PopClient.TraceOptions Enumeration

Specifies the logging options that the PopClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum PopClient.TraceOptions
```

[C#]

```
[Flags]
public enum PopClient.TraceOptions
```

Remarks

The PopClient class uses the **TraceOptions** enumeration to specify what kind of debugging information is written to the trace logfile. These options are only meaningful when trace logging is enabled by setting the **Trace** property to **true**.

Members

Member Name	Description	Value
traceDefault	The default trace logging option. This is the same as specifying the **traceInfo** option.	0
traceInfo	All network function calls are written to the trace file. This is the default value.	0
traceError	Only those network function calls which fail are recorded in the trace file.	1
traceWarning	Only those network function calls which fail, or return values which indicate a warning, are recorded in the trace file.	2
traceHexDump	All network function calls are written to the trace file, plus all the data that is sent or received is displayed, in both ASCII and hexadecimal format.	4
traceProcess	All function calls in the current process are logged, rather than only those functions in the current thread. This option is useful for multithreaded applications that are using worker threads.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.PopClient (in SocketTools.PopClient.dll)

See Also

SocketTools Namespace

PopClient.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub PopClient.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void PopClient.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender
The source of the event.

e
An ErrorEventArgs that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.PopClient (in SocketTools.PopClient.dll)

See Also

SocketTools Namespace

PopClient.OnProgressEventHandler Delegate

Represents the method that will handle the OnProgress event.

[Visual Basic]

```
Public Delegate Sub PopClient.OnProgressEventHandler( _
   ByVal sender As Object, _
   ByVal e As ProgressEventArgs _
)
```

[C#]

```
public delegate void PopClient.OnProgressEventHandler(
      object sender,
      ProgressEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A ProgressEventArgs that contains the event data.

Remarks

When you create an **OnProgressEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnProgressEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.PopClient (in SocketTools.PopClient.dll)

See Also

SocketTools Namespace

PopClientException Class

The exception that is thrown when a client error occurs.

For a list of all members of this type, see PopClientException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.PopClientException

[Visual Basic]

```
Public Class PopClientException
    Inherits ApplicationException
```

[C#]

```
public class PopClientException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A PopClientException is thrown by the PopClient class when an error occurs.

The default constructor for the PopClientException class sets the **ErrorCode** property to the last client error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.PopClient (in SocketTools.PopClient.dll)

See Also

PopClientException Members | SocketTools Namespace

PopClientException Members

PopClientException overview

Public Instance Constructors

PopClientException	Overloaded. Initializes a new instance of the PopClientException class.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

PopClientException Class | SocketTools Namespace

PopClientException Constructor

Initializes a new instance of the PopClientException class with the last network error code.

Overload List

Initializes a new instance of the PopClientException class with the last network error code.

```
public PopClientException();
```

Initializes a new instance of the PopClientException class with a specified error number.

```
public PopClientException(int);
```

Initializes a new instance of the PopClientException class with a specified error message.

```
public PopClientException(string);
```

Initializes a new instance of the PopClientException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
public PopClientException(string,Exception);
```

See Also

PopClientException Class | SocketTools Namespace

PopClientException Constructor ()

Initializes a new instance of the PopClientException class with the last network error code.

[Visual Basic]
```
Overloads Public Sub New()
```

[C#]
```
public PopClientException();
```

Remarks

The ctor constructor sets the **ErrorCode** property to the last client error that occurred. For more information about the errors that may occur, refer to the PopClient.ErrorCode enumeration.

See Also

PopClientException Class | SocketTools Namespace | PopClientException Constructor Overload List

PopClientException Constructor (String)

Initializes a new instance of the PopClientException class with a specified error message.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]

```
public PopClientException(
   string message
);
```

Parameters

message

The error message that explains the reason for the exception.

Remarks

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

PopClientException Class | SocketTools Namespace | PopClientException Constructor Overload List

PopClientException Constructor (String, Exception)

Initializes a new instance of the PopClientException class with a specified error message and a reference to the inner exception that is the cause of this exception.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal innerException As Exception _
)
```

[C#]

```
public PopClientException(
   string message,
   Exception innerException
);
```

Parameters

message

The error message that explains the reason for the exception.

innerException

The exception that is the cause of the current exception. If the ***innerException*** parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

PopClientException Class | SocketTools Namespace | PopClientException Constructor Overload List

PopClientException Constructor (Int32)

Initializes a new instance of the PopClientException class with a specified error number.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

[C#]

```
public PopClientException(
   int code
);
```

Parameters

code

An integer value which specifies an error code.

Remarks

This constructor sets the **ErrorCode** property to the specified error code. For more information about the errors that may occur, refer to the PopClient.ErrorCode enumeration.

See Also

PopClientException Class | SocketTools Namespace | PopClientException Constructor Overload List

PopClientException Properties

The properties of the **PopClientException** class are listed below. For a complete list of **PopClientException** class members, see the PopClientException Members topic.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

PopClientException Class | SocketTools Namespace

PopClientException.ErrorCode Property

Gets a value which specifies the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property ErrorCode As ErrorCode
```

[C#]

```
public PopClient.ErrorCode ErrorCode {get;}
```

Property Value

Returns a PopClient.ErrorCode enumeration value which specifies the error code.

Remarks

The **ErrorCode** property returns the error code that specifies the cause of the exception.

The default constructor for the PopClientException class sets the error code to the last network error that occurred. For more information about the errors that may occur, refer to the PopClient.ErrorCode enumeration.

See Also

PopClientException Class | SocketTools Namespace

PopClientException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]

```
Overrides Public ReadOnly Property Message As String
```

[C#]

```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

PopClientException Class | SocketTools Namespace

PopClientException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception. This value is the same as the values returned by the Windows Sockets API. For more information about socket error codes, see the Windows Socket Version 2 API error code documentation in MSDN.

See Also

PopClientException Class | SocketTools Namespace

PopClientException Methods

The methods of the **PopClientException** class are listed below. For a complete list of **PopClientException** class members, see the PopClientException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

PopClientException Class | SocketTools Namespace

PopClientException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]
```
Overrides Public Function ToString() As String
```

[C#]
```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

PopClientException Class | SocketTools Namespace

PopClient.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see PopClient.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.PopClient.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class PopClient.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class PopClient.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the PopClient class.

Example

```
<Assembly: SocketTools.PopClient.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.PopClient.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.PopClient (in SocketTools.PopClient.dll)

See Also

PopClient.RuntimeLicenseAttribute Members | SocketTools Namespace

PopClient.RuntimeLicenseAttribute Members

PopClient.RuntimeLicenseAttribute overview

Public Instance Constructors

PopClient.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

PopClient.RuntimeLicenseAttribute Class | SocketTools Namespace

PopClient.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public PopClient.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the PopClient class.

See Also

PopClient.RuntimeLicenseAttribute Class | SocketTools Namespace

PopClient.RuntimeLicenseAttribute Properties

The properties of the **PopClient.RuntimeLicenseAttribute** class are listed below. For a complete list of **PopClient.RuntimeLicenseAttribute** class members, see the PopClient.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

PopClient.RuntimeLicenseAttribute Class | SocketTools Namespace

PopClient.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]
```
Public Property LicenseKey As String
```

[C#]
```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

PopClient.RuntimeLicenseAttribute Class | SocketTools Namespace

RshClient Class

Implements the Remote Shell and Remote Login protocols.

For a list of all members of this type, see RshClient Members.

System.Object
  SocketTools.RshClient

[Visual Basic]
```
Public Class RshClient
    Implements IDisposable
```

[C#]
```
public class RshClient : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RshClient class is used to execute a command on a server and return the output of that command to the client. This is most commonly used with UNIX based servers, although there are implementations of remote command servers for the Windows operating system. The class supports both the rsh and rshell remote execution protocols and provides functions which can be used to search the data stream for specific sequences of characters. This makes it extremely easy to write Windows applications which serve as light-weight client interfaces to commands being executed on a UNIX server or another Windows system. The class can also be used to establish a remote terminal session using the rlogin protocol, which is similar to how the Telnet protocol functions.

Requirements

Namespace: SocketTools

Assembly: SocketTools.RshClient (in SocketTools.RshClient.dll)

See Also

RshClient Members | SocketTools Namespace

RshClient Members

RshClient overview

Public Static (Shared) Fields

rshPortExecute	A constant value which specifies the default port number for the rexec service.
rshPortLogin	A constant value which specifies the default port number for the rlogin service.
rshPortShell	A constant value which specifies the default port number for the rshell service.
rshTimeout	A constant value which specifies the default timeout period.

Public Instance Constructors

RshClient Constructor	Initializes a new instance of the RshClient class.

Public Instance Properties

AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
Blocking	Gets and sets a value which indicates if the client is in blocking mode.
CodePage	Gets and sets the code page used when reading and writing text.
Command	Gets and sets a value which specifies the command to be executed on the remote host.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsConnected	Gets a value which indicates if a connection to the remote host has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsReadable	Gets a value which indicates if there is data available to be read from the socket connection to the server.
IsWritable	Gets a value which indicates if data can be written to the client without blocking.
LastError	Gets and sets a value which specifies the last error that has occurred.

LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets the local Internet address that the client is bound to.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
Options	Gets and sets a value which specifies one or more client options.
Password	Gets and sets the password used to authenticate the client session.
RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.
Status	Gets a value which specifies the current status of the client.
Terminal	Gets and sets the terminal type used for a remote login session.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
UserName	Gets and sets the username used to authenticate the client session.
Version	Gets a value which returns the current version of the RshClient class library.

Public Instance Methods

AttachThread	Attach an instance of the class to the current thread
Cancel	Cancel the current blocking client operation.
Disconnect	Terminate the connection with a remote host.

Dispose	Overloaded. Releases all resources used by RshClient.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
Execute	Overloaded. Execute the specified command on the remote host.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the RshClient class.
Login	Overloaded. Establish an interactive terminal session for the specified user.
Read	Overloaded. Read data from the server and store it in a byte array.
Reset	Reset the internal state of the object, resetting all properties to their default values.
Search	Overloaded. Search for a specific character sequence in the data stream.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
Write	Overloaded. Write one or more bytes of data to the server.

Public Instance Events

OnCancel	Occurs when a blocking client operation is canceled.
OnConnect	Occurs when a connection is established with the remote host.
OnDisconnect	Occurs when the remote host disconnects from the local system.
OnError	Occurs when an client operation fails.
OnRead	Occurs when data is available to be read from the client.
OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
OnWrite	Occurs when data can be written to the client.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the RshClient class and optionally

	releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

RshClient Class | SocketTools Namespace

RshClient Constructor

Initializes a new instance of the RshClient class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public RshClient();
```

See Also

RshClient Class | SocketTools Namespace

RshClient Properties

The properties of the **RshClient** class are listed below. For a complete list of **RshClient** class members, see the RshClient Members topic.

Public Instance Properties

AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
Blocking	Gets and sets a value which indicates if the client is in blocking mode.
CodePage	Gets and sets the code page used when reading and writing text.
Command	Gets and sets a value which specifies the command to be executed on the remote host.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsConnected	Gets a value which indicates if a connection to the remote host has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsReadable	Gets a value which indicates if there is data available to be read from the socket connection to the server.
IsWritable	Gets a value which indicates if data can be written to the client without blocking.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets the local Internet address that the client is bound to.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
Options	Gets and sets a value which specifies one or more client options.
Password	Gets and sets the password used to authenticate the client session.

RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.
Status	Gets a value which specifies the current status of the client.
Terminal	Gets and sets the terminal type used for a remote login session.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
UserName	Gets and sets the username used to authenticate the client session.
Version	Gets a value which returns the current version of the RshClient class library.

See Also

RshClient Class | SocketTools Namespace

RshClient.AutoResolve Property

Gets and sets a value that determines if host names and addresses are automatically resolved.

[Visual Basic]
```
Public Property AutoResolve As Boolean
```

[C#]
```
public bool AutoResolve {get; set;}
```

Property Value

Returns **true** if host names are automatically resolved to Internet addresses. The default value is **false**.

Remarks

Setting the **AutoResolve** property determines if the class automatically resolves host names and addresses specified by the **HostName** and **HostAddress** properties. If set to **true**, setting the **HostName** property will cause the class to automatically determine the corresponding IP address and update the **HostAddress** property accordingly. Likewise, setting the **HostAddress** property will cause the class to determine the host name and update the **HostName** property. Setting this property to **false** prevents the class from resolving host names until a connection attempt is made.

It is important to note that setting the **HostName** or **HostAddress** property may cause the current thread to block, sometimes for several seconds, until the name or address is resolved. To prevent this behavior, set this property value to **false**.

See Also

RshClient Class | SocketTools Namespace

RshClient.Blocking Property

Gets and sets a value which indicates if the client is in blocking mode.

[Visual Basic]

```
Public Property Blocking As Boolean
```

[C#]

```
public bool Blocking {get; set;}
```

Property Value

Returns **true** if the client is in blocking mode; otherwise it returns **false**. The default value is **true**.

Remarks

Setting the **Blocking** property determines if client operations complete synchronously or asynchronously. If set to **true**, then each client operation (such as sending or receiving data) will return when the operation has completed or timed-out. If set to **false**, client operations will return immediately. If the operation would result in the client blocking (such as attempting to read data when no data has been sent by the remote host), an error is generated.

It is important to note that certain events, such as **OnDisconnect, OnRead** and **OnWrite** are only fired if the client is in non-blocking mode.

See Also

RshClient Class | SocketTools Namespace

RshClient.CodePage Property

Gets and sets the code page used when reading and writing text.

[Visual Basic]

```
Public Property CodePage As Integer
```

[C#]

```
public int CodePage {get; set;}
```

Property Value

An integer value which specifies the current code page. A value of zero specifies the default code page for the current locale should be used. To preserve the original Unicode text, you can use code page 65001 which specifies UTF-8 character encoding.

Remarks

All data which is exchanged over a socket is sent and received as 8-bit bytes, typically referred to as "octets" in networking terminology. However, strings in .NET are Unicode where each character is represented by 16 bits. To send and receive data using strings, these Unicode strings are converted to a stream of bytes.

By default, strings are converted to an array of bytes using the code page for the current locale, mapping the 16-bit Unicode characters to bytes. Similarly, when reading data from the socket into a string buffer, the stream of bytes received from the remote host are converted to Unicode before they are returned to your application.

If you are exchanging text with another system and it appears to corrupted or characters are being replaced with question marks or other symbols, it is likely the system is sending text which is using a different character encoding. Most services use UTF-8 encoding to represent non-ASCII characters and selecting the UTF-8 code page will typically resolve the issue.

Strings are only guaranteed to be safe when sending and receiving text. Using a string data type is not recommended when reading or writing binary data to a socket. If possible, you should always use a byte array as the buffer parameter for the Read and Write methods whenever you are exchanging binary data.

For backwards compatibility, this class defaults to using the code page for the current locale. This property value directly corresponds to Windows code page identifiers, and will accept any valid code page supported by the .NET Framework. Setting this property to an invalid code page will generate an exception.

See Also

RshClient Class | SocketTools Namespace

RshClient.HostAddress Property

Gets and sets a value which specifies the Internet address used to establish a connection.

[Visual Basic]

```
Public Property HostAddress As String
```

[C#]

```
public string HostAddress {get; set;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **HostAddress** property can be used to set the Internet address for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the address is assigned to a valid host name, the **HostName** property will be updated with that value.

See Also

RshClient Class | SocketTools Namespace

RshClient.HostName Property

Gets and sets a value which specifies the host name used to establish a connection.

[Visual Basic]

```
Public Property HostName As String
```

[C#]

```
public string HostName {get; set;}
```

Property Value

A string which specifies a host name.

Remarks

The **HostName** property can be used to set the host name for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the name can be resolved to a valid Internet address, the **HostAddress** property will be updated with that value.

See Also

RshClient Class | SocketTools Namespace

RshClient.IsBlocked Property

Gets a value which indicates if the current thread is performing a blocking client operation.

[Visual Basic]
```
Public ReadOnly Property IsBlocked As Boolean
```

[C#]
```
public bool IsBlocked {get;}
```

Property Value

Returns **true** if the current thread is blocking, otherwise returns **false**.

Remarks

The **IsBlocked** property returns **true** if the current thread is blocked performing an operation. Because the Windows Sockets API only permits one blocking operation per thread of execution, this property should be checked before starting any blocking operation in response to an event.

If the **IsBlocked** property returns **false**, this means there are no blocking operations on the current thread at that time. However, this does not guarantee that the next client operation will not fail. An application should always check the return value from a client operation and check the value of the **LastError** property if an error occurs.

Note that this property will return **true** if there is any blocking operation being performed by the current thread, regardless of whether this specific instance of the class is responsible for the blocking operation or not.

See Also

RshClient Class | SocketTools Namespace

RshClient.IsConnected Property

Gets a value which indicates if a connection to the remote host has been established.

[Visual Basic]

```
Public ReadOnly Property IsConnected As Boolean
```

[C#]

```
public bool IsConnected {get;}
```

Property Value

Returns **true** if the connection has been established; otherwise returns **false**.

Remarks

The **IsConnected** property can only be used to indicate if there is still a logical connection to the remote host. It cannot be used to detect abnormal conditions such as the remote host aborting the connection, the physical network connection being lost or other critical errors.

See Also

RshClient Class | SocketTools Namespace

RshClient.IsInitialized Property

Gets a value which indicates if the current instance of the class has been initialized successfully.

[Visual Basic]

```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]

```
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the class instance has been initialized; otherwise returns **false**.

Remarks

The **IsInitialized** property is used to determine if the current instance of the class has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the class may not be able to initialize itself.

The most common reasons that a class instance may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required networking libraries or not being able to access the system registry.

See Also

RshClient Class | SocketTools Namespace

RshClient.IsReadable Property

Gets a value which indicates if there is data available to be read from the socket connection to the server.

[Visual Basic]

```
Public ReadOnly Property IsReadable As Boolean
```

[C#]

```
public bool IsReadable {get;}
```

Property Value

Returns **true** if there is data available to be read; otherwise returns **false**.

Remarks

The **IsReadable** property returns **true** if data can be read from the client without blocking. For non-blocking sessions, this property can be checked before the application attempts to read the client. Note that even if this property does return **true** indicating that there is data available to be read, applications should always check the return value from the **Read** method.

See Also

RshClient Class | SocketTools Namespace

RshClient.IsWritable Property

Gets a value which indicates if data can be written to the client without blocking.

[Visual Basic]

```
Public ReadOnly Property IsWritable As Boolean
```

[C#]

```
public bool IsWritable {get;}
```

Property Value

Returns **true** if data can be written to the client; otherwise returns **false**.

Remarks

The **IsWritable** property returns **true** if data can be written to the client without blocking. For non-blocking sessions, this property can be checked before the application attempts to write data to the client. Note that even if this property does return **true** indicating that data can be written to the client, applications should always check the return value from the **Write** method.

See Also

RshClient Class | SocketTools Namespace

RshClient.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]
```
Public Property LastError As ErrorCode
```

[C#]
```
public RshClient.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

RshClient Class | SocketTools Namespace

RshClient.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

RshClient Class | SocketTools Namespace

RshClient.LocalAddress Property

Gets the local Internet address that the client is bound to.

[Visual Basic]
```
Public ReadOnly Property LocalAddress As String
```

[C#]
```
public string LocalAddress {get;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **LocalAddress** property returns the local Internet address that the client is bound to when a connection is established with a remote host. This property may return either an IPv4 or IPv6 formatted address, depending on the type of connection that was established.

See Also

RshClient Class | SocketTools Namespace

RshClient.LocalName Property

Gets a value which specifies the host name for the local system.

[Visual Basic]
```
Public ReadOnly Property LocalName As String
```

[C#]
```
public string LocalName {get;}
```

Property Value

A string which specifies the local host name.

Remarks

The **LocalName** property returns the fully-qualified host name assigned to the local system. If the system has not been configured with an Internet domain name, then this property will return the NetBIOS name assigned to the local system.

See Also

RshClient Class | SocketTools Namespace

RshClient.LocalPort Property

Gets the local port number the client is bound to.

[Visual Basic]
```
Public ReadOnly Property LocalPort As Integer
```

[C#]
```
public int LocalPort {get;}
```

Property Value

An integer value which specifies a port number. The default value is 0.

Remarks

The **LocalPort** property is used to identify the local port number that the client is bound to to when a connection is established with a remote host.

See Also

RshClient Class | SocketTools Namespace

RshClient.Options Property

Gets and sets a value which specifies one or more client options.

[Visual Basic]

```
Public Property Options As RemoteOptions
```

[C#]

```
public RshClient.RemoteOptions Options {get; set;}
```

Property Value

Returns one or more RemoteOptions enumeration flags which specify the options for the client. The default value for this property is **rshOptionNone**.

Remarks

The **Options** property specifies one or more default options options which are used when establishing a connection using the **Connect** method.

See Also

RshClient Class | SocketTools Namespace

RshClient.Password Property

Gets and sets the password used to authenticate the client session.

[Visual Basic]
```
Public Property Password As String
```

[C#]
```
public string Password {get; set;}
```

Property Value

A string which specifies the password.

Remarks

If a password is not specified when the **Execute** method is called, the value of this property will be used as the default password when establishing a connection with the server.

See Also

RshClient Class | SocketTools Namespace

RshClient.RemotePort Property

Gets and sets a value which specifies the remote port number.

[Visual Basic]

```
Public Property RemotePort As Integer
```

[C#]

```
public int RemotePort {get; set;}
```

Property Value

An integer value which specifies a port number.

Remarks

The **RemotePort** property is used to set the port number that will be used to establish a connection with a remote host. If the port number specifies a well-known port, the **RemoteService** property will be updated with that name.

See Also

RshClient Class | SocketTools Namespace

RshClient.Status Property

Gets a value which specifies the current status of the client.

[Visual Basic]

```
Public ReadOnly Property Status As RemoteStatus
```

[C#]

```
public RshClient.RemoteStatus Status {get;}
```

Property Value

A RemoteStatus enumeration value which specifies the current client status.

Remarks

The **Status** property returns the current status of the client. This property can be used to check on blocking connections to determine if the client is interacting with the remote host before taking some action.

See Also

RshClient Class | SocketTools Namespace

RshClient.Terminal Property

Gets and sets the terminal type used for a remote login session.

[Visual Basic]

```
Public Property Terminal As String
```

[C#]

```
public string Terminal {get; set;}
```

Property Value

A string which specifies the terminal type.

Remarks

The **Terminal** property specifies the terminal type of the remote host for display purposes. On UNIX based systems, the terminal name corresponds to a termcap or terminfo entry as set in the TERM environment variable. On Windows based systems which implement the rlogin service, this property may be ignored and the server will assume that the client is capable of displaying ANSI escape sequences. On VMS systems, the terminal name should correspond to the terminal type used with the SET TERMINAL/DEVICE command.

If this property is set to an empty string and no terminal type is specified when the **Login** method is called, a default terminal type named "unknown" will be used. On most UNIX and VMS systems this defines a terminal which is not capable of cursor positioning using control or escape sequences. This terminal type may not be recognized and an error may be displayed when the user logs in indicating that the terminal type is invalid.

Refer to the documentation for the server system to determine what terminal type names are available to you. Remember that on UNIX systems, the terminal type is case-sensitive. Some of the more common terminal types are:

Terminal	Description
ansi	This terminal type is usually available on UNIX based servers. This specifies that the client is capable of displaying standard ANSI escape sequences for cursor control.
dumb	This terminal type typically specifies a terminal display which does not support control or escape sequences for cursor positioning. If you do not want escape sequences embedded in the data stream and the server returns an error if the terminal type is not specified, try using this terminal type.
pcansi	This terminal type is usually available on UNIX based servers. This specifies that the client is a using a PC terminal emulator that supports basic ANSI escape sequences for cursor control. This may also enable escape sequences which can set the display colors.
vt100	This terminal type is usually available on UNIX and VMS based servers. On some VMS systems this

	string may need to be specified as DEC-VT100. This specifies that the client is capable of emulating a DEC VT100 terminal. The VT100 supports many of the same cursor control sequences as an ANSI terminal.
vt220	This terminal type is usually available on UNIX and VMS based servers. On some VMS systems this string may need to be specified as DEC-VT220. This specifies that the client is capable of emulating a DEC VT220 terminal, which is a later version of the VT100.
vt320	This terminal type is usually available on UNIX and VMS based servers. On some VMS systems this string may need to be specified as DEC-VT320. This specifies that the client is capable of emulating a DEC VT320 terminal, which is similar to the VT100 and VT220 and provides advanced features such as the ability to set display colors.
xterm	This terminal type is may be available on UNIX based servers which have X Windows installed. This specifies that the client is a using the X Windows xterm emulator which supports standard ANSI escape sequences for cursor control.

See Also

RshClient Class | SocketTools Namespace

RshClient.ThreadModel Property

Gets and sets a value which specifies the threading model for the class instance.

[Visual Basic]
```
Public Property ThreadModel As ThreadingModel
```

[C#]
```
public RshClient.ThreadingModel ThreadModel {get; set;}
```

Property Value

Returns one or more ThreadingModel enumeration value which specifies the threading model for the client. The default value for this property is **modelSingleThread**.

Remarks

The **ThreadModel** property specifies the threading model that is used by the class instance when a connection is established. The default value for this property is **modelSingleThread**, which specifies that only the thread that established the connection should be permitted to invoke methods. It is important to note that this threading model does not limit the application to a single thread of execution. When a session is established using the **Connect** method, that session is attached to the thread that created it. From that point on, until the session is terminated, only the owner may invoke methods in that instance of the class. The ownership of the class instance may be transferred from one thread to another using the **AttachThread** method.

Setting this property to **modelFreeThread** disables certain internal safety checks that are performed by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads. The application must ensure that no two threads will attempt to invoke a blocking method at the same time. In other words, if one thread invokes a method, the application must ensure that another thread will not attempt to invoke any other method at the same time using the same instance of the class.

Changing the value of this property will not affect an active client session. The threading model must be specified prior to invoking the **Connect** method.

See Also

RshClient Class | SocketTools Namespace | AttachThread Method | ThreadingModel Enumeration | ThreadModel Attribute

RshClient.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]
```
Public Property ThrowError As Boolean
```

[C#]
```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

RshClient Class | SocketTools Namespace

RshClient.Timeout Property

Gets and sets a value which specifies a timeout period in seconds.

[Visual Basic]

```
Public Property Timeout As Integer
```

[C#]

```
public int Timeout {get; set;}
```

Property Value

An integer value which specifies a timeout period in seconds.

Remarks

Setting the **Timeout** property specifies the number of seconds until a blocking operation fails and returns an error.

The timeout period is only used when the client is in blocking mode. Although this property can be changed when the client is in non-blocking mode, the value will be ignored until the client is returned to blocking mode.

For most applications it is recommended the timeout period be set between 10 and 20 seconds.

See Also

RshClient Class | SocketTools Namespace

RshClient.Trace Property

Gets and sets a value which indicates if network function logging is enabled.

[Visual Basic]

```
Public Property Trace As Boolean
```

[C#]

```
public bool Trace {get; set;}
```

Property Value

Returns **true** if network function tracing is enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Trace** property is used to enable (or disable) the tracing of network function calls. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the SocketTools networking classes and is enabled or disabled for an entire process. This means that once trace logging is enabled for a given component, all of the function calls made by the process using any of the SocketTools classes will be logged. For example, if you have an application using both the File Transfer Protocol and Post Office Protocol classes, and you set the **Trace** property to **true**, function calls made by both classes will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all classes used by the process.

If trace logging is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the logfile is fairly large. Since logfiles can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

When redistributing your application, make sure that you include the **SocketTools10.TraceLog.dll** module with your installation. If this library is not present, then no trace output will be generated and the value of the **Trace** property will be ignored. Only those function calls made by the SocketTools networking classes will be logged. Calls made directly to the Windows Sockets API, or calls made by other classes, will not be logged.

See Also

RshClient Class | SocketTools Namespace

RshClient.TraceFile Property

Gets and sets a value which specifies the name of the logfile.

[Visual Basic]

```
Public Property TraceFile As String
```

[C#]

```
public string TraceFile {get; set;}
```

Property Value

A string which specifies the name of the file.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named **SocketTools.log** is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFile** property should be set to the same value for each instance. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
 MyApp INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0 MyApp WRN:
connect(46, 192.0.0.1:1234, 16) returned -1 [10035] MyApp ERR: accept(46,
NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the control is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Note that if the specified file cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

RshClient Class | SocketTools Namespace

RshClient.TraceFlags Property

Gets and sets a value which specifies the client function tracing flags.

[Visual Basic]

```
Public Property TraceFlags As TraceOptions
```

[C#]

```
public RshClient.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration which specifies the amount of detail written to the trace logfile.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Because network function tracing is enabled per-process, the trace flags are shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFlags** property should be set to the same value for each instance. Changing the trace flags for any one instance of the class will affect the logging performed for all SocketTools classes used by the application.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being written and the error **errorOperationWouldBlock** occurs, a warning is generated because the application simply needs to attempt to write the data at a later time.

See Also

RshClient Class | SocketTools Namespace

RshClient.UserName Property

Gets and sets the username used to authenticate the client session.

[Visual Basic]

```
Public Property UserName As String
```

[C#]

```
public string UserName {get; set;}
```

Property Value

A string which specifies the username.

Remarks

If a username is not specified when the **Execute** or **Login** method is called, the value of this property will be used as the default username when establishing a connection with the server.

See Also

RshClient Class | SocketTools Namespace

RshClient.Version Property

Gets a value which returns the current version of the RshClient class library.

[Visual Basic]
```
Public ReadOnly Property Version As String
```

[C#]
```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the RshClient class library. This value can be used by an application for validation and debugging purposes.

See Also

RshClient Class | SocketTools Namespace

RshClient Methods

The methods of the **RshClient** class are listed below. For a complete list of **RshClient** class members, see the RshClient Members topic.

Public Instance Methods

AttachThread	Attach an instance of the class to the current thread
Cancel	Cancel the current blocking client operation.
Disconnect	Terminate the connection with a remote host.
Dispose	Overloaded. Releases all resources used by RshClient.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
Execute	Overloaded. Execute the specified command on the remote host.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the RshClient class.
Login	Overloaded. Establish an interactive terminal session for the specified user.
Read	Overloaded. Read data from the server and store it in a byte array.
Reset	Reset the internal state of the object, resetting all properties to their default values.
Search	Overloaded. Search for a specific character sequence in the data stream.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
Write	Overloaded. Write one or more bytes of data to the server.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the RshClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.

MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

RshClient Class | SocketTools Namespace

RshClient.AttachThread Method

Attach an instance of the class to the current thread

[Visual Basic]
```
Public Function AttachThread() As Boolean
```

[C#]
```
public bool AttachThread();
```

Return Value

A boolean value which specifies if the client could be attached to the current thread. If this method returns **false**, the client could not be attached to the thread and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

When an instance of the class is created it is associated with the current thread that created it. Normally, if another thread attempts to perform an operation using that instance, an error is returned. This is used to ensure that other threads cannot interfere with an operation being performed by the owner thread. In some cases, it may be desirable for one thread in an application to create an instance of the class, establish a connection and then pass that instance to another worker thread. The **AttachThread** method can be used to change the ownership of the class instance to the new worker thread.

This method should be called by the new thread immediately after it has been created, and if the new thread does not release the handle itself, the ownership of the handle should be restored by the original thread. Under no circumstances should **AttachThread** be used to forcibly destroy an instance of a class allocated by another thread while a blocking operation is in progress. To cancel a blocking operation, use the **Cancel** method and then delete the class instance after the blocking function exits and control is returned to the current thread.

See Also

RshClient Class | SocketTools Namespace

RshClient.Cancel Method

Cancel the current blocking client operation.

[Visual Basic]

```
Public Sub Cancel()
```

[C#]

```
public void Cancel();
```

Remarks

When the **Cancel** method is called, the blocking client operation will not immediately fail. An internal flag is set which causes the blocking operation to exit with an error. This means that the application cannot cancel an operation and immediately perform some other blocking function. Instead it must allow the calling stack to unwind, returning back to the blocking operation before making any further function calls.

See Also

RshClient Class | SocketTools Namespace

RshClient.Disconnect Method

Terminate the connection with a remote host.

[Visual Basic]

```
Public Sub Disconnect()
```

[C#]

```
public void Disconnect();
```

Remarks

The **Disconnect** method terminates the connection with the remote host and releases the client handle allocated by the class. Note that the socket is not immediately released when the connection is terminated and will enter a wait state for two minutes. After the time wait period has elapsed, the client will be released by the operating system. This is a normal safety mechanism to handle any packets that may arrive after the connection has been closed.

See Also

RshClient Class | SocketTools Namespace

RshClient.Dispose Method

Releases all resources used by RshClient.

Overload List

Releases all resources used by RshClient.

```
public void Dispose();
```

Releases the unmanaged resources allocated by the RshClient class and optionally releases the managed resources.

```
protected virtual void Dispose(bool);
```

See Also

RshClient Class | SocketTools Namespace

RshClient.Dispose Method ()

Releases all resources used by RshClient.

[Visual Basic]

```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]

```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

RshClient Class | SocketTools Namespace | RshClient.Dispose Overload List

RshClient.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the RshClient class and optionally releases the managed resources.

[Visual Basic]

```
Overridable Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]

```
protected virtual void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **RshClient** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

RshClient Class | SocketTools Namespace | RshClient.Dispose Overload List

RshClient.Execute Method

Execute the specified command on the remote host.

Overload List

Execute the specified command on the remote host.

public bool Execute();

Execute the specified command on the remote host.

public bool Execute(string);

Execute the specified command on the remote host.

public bool Execute(string,int,string);

Execute the specified command on the remote host.

public bool Execute(string,int,string,string,string);

Execute the specified command on the remote host.

public bool Execute(string,int,string,string,string,int);

Execute the specified command on the remote host.

public bool Execute(string,int,string,string,string,int,RemoteOptions);

Execute the specified command on the remote host.

public bool Execute(string,string);

Execute the specified command on the remote host.

public bool Execute(string,string,string);

Execute the specified command on the remote host.

public bool Execute(string,string,string,string);

See Also

RshClient Class | SocketTools Namespace

RshClient.Execute Method (String, Int32, String)

Execute the specified command on the remote host.

[Visual Basic]

```
Overloads Public Function Execute( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal command As String _
) As Boolean
```

[C#]

```
public bool Execute(
   string hostName,
   int hostPort,
   string command
);
```

Parameters

hostName

A string which specifies the name of the server to connect to. The string may either be an IP address or a fully qualified domain name.

hostPort

An integer value which specifies the port number to connect to. This method may be used to either connect to the rexec service or the rshell service, and which service is selected depends on the port number provided. One of the following values should be used:

Port	Description
rshPortExec	A connection is established with the server using port 512, the rexec service. This service requires that the client provide a username and password to execute the specified command.
rshPortShell	A connection is established with the server using port 514, the rshell service. This service uses host equivalence to authenticate the user. With host equivalence, the remote server considers the client to be equivalent to itself, and as long as the specified user exists on the remote host, the client is permitted to execute commands on behalf of the user without requiring a password. Host equivalence is configured by the server administrator.

command

An string which specifies the command to be executed on the server.

Return Value

A boolean value which specifies if the connection has been established. If the socket is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the socket has been successfully created and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Execute** method executes the specified command on a remote host. Output from the command may be read using the **Read** method. Input can be supplied to the program using the **Write** method. To search for a specific sequence of bytes in the output returned by the server, use the **Search** method.

The value of the **UserName** and **Password** properties specify the username and password that will be used to authenticate the client session. The value of the **Timeout** property specifies the timeout period. The value of the **Options** property specifies the options that will be used when establishing the connection.

See Also

RshClient Class | SocketTools Namespace | RshClient.Execute Overload List

RshClient.Execute Method (String, Int32, String, String, String)

Execute the specified command on the remote host.

[Visual Basic]

```
Overloads Public Function Execute( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal command As String _
) As Boolean
```

[C#]

```
public bool Execute(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   string command
);
```

Parameters

hostName

A string which specifies the name of the server to connect to. The string may either be an IP address or a fully qualified domain name.

hostPort

An integer value which specifies the port number to connect to. This method may be used to either connect to the rexec service or the rshell service, and which service is selected depends on the port number provided. One of the following values should be used:

Port	Description
rshPortExec	A connection is established with the server using port 512, the rexec service. This service requires that the client provide a username and password to execute the specified command.
rshPortShell	A connection is established with the server using port 514, the rshell service. This service uses host equivalence to authenticate the user. With host equivalence, the remote server considers the client to be equivalent to itself, and as long as the specified user exists on the remote host, the client is permitted to execute commands on behalf of the user without requiring a password. Host equivalence is configured by the server administrator.

userName

A string which specifies the username which used to authenticate the client session.

userPassword

A string which specifies the password to be used to authenticate the user. A password is only used if the client is connecting to the rexec service. The rshell service uses host equivalence to authenticate the user and this argument will be ignored.

command

An string which specifies the command to be executed on the server.

Return Value

A boolean value which specifies if the connection has been established. If the socket is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the socket has been successfully created and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Execute** method executes the specified command on a remote host. Output from the command may be read using the **Read** method. Input can be supplied to the program using the **Write** method. To search for a specific sequence of bytes in the output returned by the server, use the **Search** method.

The value of the **Timeout** property specifies the timeout period. The value of the **Options** property specifies the options that will be used when establishing the connection.

See Also

RshClient Class | SocketTools Namespace | RshClient.Execute Overload List

RshClient.Execute Method (String, Int32, String, String, String, Int32)

Execute the specified command on the remote host.

```
[Visual Basic]
Overloads Public Function Execute( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal command As String, _
   ByVal timeout As Integer _
) As Boolean
```

```
[C#]
public bool Execute(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   string command,
   int timeout
);
```

Parameters

hostName

A string which specifies the name of the server to connect to. The string may either be an IP address or a fully qualified domain name.

hostPort

An integer value which specifies the port number to connect to. This method may be used to either connect to the rexec service or the rshell service, and which service is selected depends on the port number provided. One of the following values should be used:

Port	Description
rshPortExec	A connection is established with the server using port 512, the rexec service. This service requires that the client provide a username and password to execute the specified command.
rshPortShell	A connection is established with the server using port 514, the rshell service. This service uses host equivalence to authenticate the user. With host equivalence, the remote server considers the client to be equivalent to itself, and as long as the specified user exists on the remote host, the client is permitted to execute commands on behalf of the user without requiring a password. Host equivalence is configured by the server administrator.

userName

A string which specifies the username which used to authenticate the client session.

userPassword

A string which specifies the password to be used to authenticate the user. A password is only used if the client is connecting to the rexec service. The rshell service uses host equivalence to authenticate the user and this argument will be ignored.

command

An string which specifies the command to be executed on the server.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

Return Value

A boolean value which specifies if the connection has been established. If the socket is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the socket has been successfully created and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Execute** method executes the specified command on a remote host. Output from the command may be read using the **Read** method. Input can be supplied to the program using the **Write** method. To search for a specific sequence of bytes in the output returned by the server, use the **Search** method.

The value of the **Options** property specifies the options that will be used when establishing the connection.

See Also

RshClient Class | SocketTools Namespace | RshClient.Execute Overload List

RshClient.Execute Method (String, Int32, String, String, String, Int32, RemoteOptions)

Execute the specified command on the remote host.

[Visual Basic]

```
Overloads Public Function Execute( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal command As String, _
   ByVal timeout As Integer, _
   ByVal options As RemoteOptions _
) As Boolean
```

[C#]

```
public bool Execute(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   string command,
   int timeout,
   RemoteOptions options
);
```

Parameters

hostName

A string which specifies the name of the server to connect to. The string may either be an IP address or a fully qualified domain name.

hostPort

An integer value which specifies the port number to connect to. This method may be used to either connect to the rexec service or the rshell service, and which service is selected depends on the port number provided. One of the following values should be used:

Port	Description
rshPortExec	A connection is established with the server using port 512, the rexec service. This service requires that the client provide a username and password to execute the specified command.
rshPortShell	A connection is established with the server using port 514, the rshell service. This service uses host equivalence to authenticate the user. With host equivalence, the remote server considers the client to be equivalent to itself, and as long as the specified user exists on the remote host, the client is permitted to execute commands on behalf of the user without requiring a password. Host equivalence is configured by the server administrator.

userName

A string which specifies the username which used to authenticate the client session.

userPassword

A string which specifies the password to be used to authenticate the user. A password is only used if the client is connecting to the rexec service. The rshell service uses host equivalence to authenticate the user and this argument will be ignored.

command

An string which specifies the command to be executed on the server.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

options

One or more of the RemoteOptions enumeration flags.

Return Value

A boolean value which specifies if the connection has been established. If the socket is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the socket has been successfully created and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Execute** method executes the specified command on a remote host. Output from the command may be read using the **Read** method. Input can be supplied to the program using the **Write** method. To search for a specific sequence of bytes in the output returned by the server, use the **Search** method.

See Also

RshClient Class | SocketTools Namespace | RshClient.Execute Overload List

RshClient.Execute Method (String, String)

Execute the specified command on the remote host.

[Visual Basic]

```
Overloads Public Function Execute( _
   ByVal hostName As String, _
   ByVal command As String _
) As Boolean
```

[C#]

```
public bool Execute(
   string hostName,
   string command
);
```

Parameters

hostName

A string which specifies the name of the server to connect to. The string may either be an IP address or a fully qualified domain name.

command

An string which specifies the command to be executed on the server.

Return Value

A boolean value which specifies if the connection has been established. If the socket is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the socket has been successfully created and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Execute** method executes the specified command on a remote host. Output from the command may be read using the **Read** method. Input can be supplied to the program using the **Write** method. To search for a specific sequence of bytes in the output returned by the server, use the **Search** method.

The value of the **RemotePort** property specifies the port number. The value of the **UserName** and **Password** properties specify the username and password that will be used to authenticate the client session. The value of the **Command** property specifies the command that will be executed. The value of the **Timeout** property specifies the timeout period. The value of the **Options** property specifies the options that will be used when establishing the connection.

See Also

RshClient Class | SocketTools Namespace | RshClient.Execute Overload List

RshClient.Execute Method (String, String, String)

Execute the specified command on the remote host.

[Visual Basic]

```
Overloads Public Function Execute( _
   ByVal hostName As String, _
   ByVal userName As String, _
   ByVal command As String _
) As Boolean
```

[C#]

```
public bool Execute(
   string hostName,
   string userName,
   string command
);
```

Parameters

hostName

A string which specifies the name of the server to connect to. The string may either be an IP address or a fully qualified domain name.

userName

A string which specifies the username which used to authenticate the client session.

command

An string which specifies the command to be executed on the server.

Return Value

A boolean value which specifies if the connection has been established. If the socket is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the socket has been successfully created and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Execute** method executes the specified command on a remote host. Output from the command may be read using the **Read** method. Input can be supplied to the program using the **Write** method. To search for a specific sequence of bytes in the output returned by the server, use the **Search** method.

The value of the **RemotePort** property specifies the remote port number. The value of the **Timeout** property specifies the timeout period. The value of the **Options** property specifies the options that will be used when establishing the connection.

See Also

RshClient Class | SocketTools Namespace | RshClient.Execute Overload List

RshClient.Execute Method (String, String, String, String)

Execute the specified command on the remote host.

[Visual Basic]

```
Overloads Public Function Execute( _
   ByVal hostName As String, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal command As String _
) As Boolean
```

[C#]

```
public bool Execute(
   string hostName,
   string userName,
   string userPassword,
   string command
);
```

Parameters

hostName

A string which specifies the name of the server to connect to. The string may either be an IP address or a fully qualified domain name.

userName

A string which specifies the username which used to authenticate the client session.

userPassword

A string which specifies the password to be used to authenticate the user. A password is only used if the client is connecting to the rexec service. The rshell service uses host equivalence to authenticate the user and this argument will be ignored.

command

An string which specifies the command to be executed on the server.

Return Value

A boolean value which specifies if the connection has been established. If the socket is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the socket has been successfully created and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Execute** method executes the specified command on a remote host. Output from the command may be read using the **Read** method. Input can be supplied to the program using the **Write** method. To search for a specific sequence of bytes in the output returned by the server, use the **Search** method.

The value of the **RemotePort** property specifies the remote port number. The value of the **Timeout** property specifies the timeout period. The value of the **Options** property specifies the options that will be used when establishing the connection.

See Also

RshClient Class | SocketTools Namespace | RshClient.Execute Overload List

RshClient.Finalize Method

Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.

[Visual Basic]

```
Overrides Protected Sub Finalize()
```

[C#]

```
protected override void Finalize();
```

See Also

RshClient Class | SocketTools Namespace

RshClient.Initialize Method

Initialize an instance of the RshClient class.

Overload List

Initialize an instance of the RshClient class.

public bool Initialize();

Initialize an instance of the RshClient class.

public bool Initialize(string);

See Also

RshClient Class | SocketTools Namespace | Uninitialize Method

RshClient.Initialize Method ()

Initialize an instance of the RshClient class.

[Visual Basic]

```
Overloads Public Function Initialize() As Boolean
```

[C#]

```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the RshClient class, loading the networking library and allocating resources for the current thread. Typically it is not necessary to explicitly call this method because the instance of the class is initialized by the class constructor. However, if the **Uninitialize** method is called, the class must be re-initialized before any other methods are called.

See Also

RshClient Class | SocketTools Namespace | RshClient.Initialize Overload List | Uninitialize Method

RshClient.Initialize Method (String)

Initialize an instance of the RshClient class.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the RshClient class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the RshClient class.

Example

The following example shows how to use the Initialize method to initialize an instance of the class. This example assumes that the license key string has been defined in code.

```
SocketTools.RshClient rshClient = new SocketTools.RshClient();

if (rshClient.Initialize(strLicenseKey) == false)
{
    MessageBox.Show(rshClient.LastErrorString, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    return;
}
```

```
Dim rshClient As New SocketTools.RshClient

If rshClient.Initialize(strLicenseKey) = False Then
    MsgBox(rshClient.LastErrorString, vbIconExclamation)
    Exit Sub
End If
```

See Also

RshClient Class | SocketTools Namespace | RshClient.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

RshClient.Login Method

Establish an interactive terminal session for the specified user.

Overload List

Establish an interactive terminal session for the specified user.

public bool Login(string);

Establish an interactive terminal session for the specified user.

public bool Login(string,int);

Establish an interactive terminal session for the specified user.

public bool Login(string,int,string);

Establish an interactive terminal session for the specified user.

public bool Login(string,int,string,int);

Establish an interactive terminal session for the specified user.

public bool Login(string,int,string,string,int);

Establish an interactive terminal session for the specified user.

public bool Login(string,int,string,string,int,RemoteOptions);

Establish an interactive terminal session for the specified user.

public bool Login(string,string);

See Also

RshClient Class | SocketTools Namespace

RshClient.Login Method (String)

Establish an interactive terminal session for the specified user.

[Visual Basic]

```
Overloads Public Function Login( _
   ByVal hostName As String _
) As Boolean
```

[C#]

```
public bool Login(
   string hostName
);
```

Parameters

hostName

A string which specifies the name of the server to connect to. The string may either be an IP address or a fully qualified domain name.

Return Value

A boolean value which specifies if the connection has been established. If the socket is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the socket has been successfully created and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Login** method logs the specified user in on the remote host. Note that no password is provided to the remote host. This is because the remote login service uses ***user equivalence***. If the client system is recognized by the remote host as being equivalent, the login will proceed directly. If the client system is not recognized, the server will prompt the user for a password. For more information about user equivalence and the remote login service, refer to your server's operating system documentation.

Output from the command may be read using the **Read** method. Input can be supplied to the program using the **Write** method. To search for a specific sequence of bytes in the output returned by the server, use the **Search** method.

See Also

RshClient Class | SocketTools Namespace | RshClient.Login Overload List

RshClient.Login Method (String, Int32)

Establish an interactive terminal session for the specified user.

[Visual Basic]
```
Overloads Public Function Login( _
   ByVal hostName As String, _
   ByVal hostPort As Integer _
) As Boolean
```

[C#]
```
public bool Login(
   string hostName,
   int hostPort
);
```

Parameters

hostName
A string which specifies the name of the server to connect to. The string may either be an IP address or a fully qualified domain name.

hostPort
An integer value which specifies the port number to connect to. The default port for this service is **rshPortLogin**.

Return Value

A boolean value which specifies if the connection has been established. If the socket is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the socket has been successfully created and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Login** method logs the specified user in on the remote host. Note that no password is provided to the remote host. This is because the remote login service uses ***user equivalence***. If the client system is recognized by the remote host as being equivalent, the login will proceed directly. If the client system is not recognized, the server will prompt the user for a password. For more information about user equivalence and the remote login service, refer to your server's operating system documentation.

Output from the command may be read using the **Read** method. Input can be supplied to the program using the **Write** method. To search for a specific sequence of bytes in the output returned by the server, use the **Search** method.

See Also

RshClient Class | SocketTools Namespace | RshClient.Login Overload List

RshClient.Login Method (String, Int32, String)

Establish an interactive terminal session for the specified user.

[Visual Basic]

```
Overloads Public Function Login( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String _
) As Boolean
```

[C#]

```
public bool Login(
   string hostName,
   int hostPort,
   string userName
);
```

Parameters

hostName

A string which specifies the name of the server to connect to. The string may either be an IP address or a fully qualified domain name.

hostPort

An integer value which specifies the port number to connect to. The default port for this service is **rshPortLogin**.

userName

A string which specifies the username which used to authenticate the client session.

Return Value

A boolean value which specifies if the connection has been established. If the socket is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the socket has been successfully created and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Login** method logs the specified user in on the remote host. Note that no password is provided to the remote host. This is because the remote login service uses ***user equivalence***. If the client system is recognized by the remote host as being equivalent, the login will proceed directly. If the client system is not recognized, the server will prompt the user for a password. For more information about user equivalence and the remote login service, refer to your server's operating system documentation.

Output from the command may be read using the **Read** method. Input can be supplied to the program using the **Write** method. To search for a specific sequence of bytes in the output returned by the server, use the **Search** method.

See Also

RshClient Class | SocketTools Namespace | RshClient.Login Overload List

RshClient.Login Method (String, Int32, String, Int32)

Establish an interactive terminal session for the specified user.

[Visual Basic]

```
Overloads Public Function Login( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Login(
   string hostName,
   int hostPort,
   string userName,
   int timeout
);
```

Parameters

hostName

A string which specifies the name of the server to connect to. The string may either be an IP address or a fully qualified domain name.

hostPort

An integer value which specifies the port number to connect to. The default port for this service is **rshPortLogin**.

userName

A string which specifies the username which used to authenticate the client session.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

Return Value

A boolean value which specifies if the connection has been established. If the socket is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the socket has been successfully created and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Login** method logs the specified user in on the remote host. Note that no password is provided to the remote host. This is because the remote login service uses ***user equivalence***. If the client system is recognized by the remote host as being equivalent, the login will proceed directly. If the client system is not recognized, the server will prompt the user for a password. For more information about user equivalence and the remote login service, refer to your server's operating system documentation.

Output from the command may be read using the **Read** method. Input can be supplied to the program using the **Write** method. To search for a specific sequence of bytes in the output returned by the server, use the **Search** method.

See Also

RshClient Class | SocketTools Namespace | RshClient.Login Overload List

RshClient.Login Method (String, Int32, String, String, Int32)

Establish an interactive terminal session for the specified user.

[Visual Basic]

```
Overloads Public Function Login( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userTerminal As String, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Login(
   string hostName,
   int hostPort,
   string userName,
   string userTerminal,
   int timeout
);
```

Parameters

hostName
A string which specifies the name of the server to connect to. The string may either be an IP address or a fully qualified domain name.

hostPort
An integer value which specifies the port number to connect to. The default port for this service is **rshPortLogin**.

userName
A string which specifies the username which used to authenticate the client session.

userTerminal
A string which specifies the client terminal type display purposes.

timeout
An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

Return Value

A boolean value which specifies if the connection has been established. If the socket is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the socket has been successfully created and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Login** method logs the specified user in on the remote host. Note that no password is provided to the remote host. This is because the remote login service uses ***user equivalence***. If the client system is recognized by the remote host as being equivalent, the login will proceed directly. If the client system is not recognized, the server will prompt the user for a password. For more information about user equivalence and the remote login service, refer to your server's operating system documentation.

Output from the command may be read using the **Read** method. Input can be supplied to the program using the **Write** method. To search for a specific sequence of bytes in the output returned by the server, use the **Search** method.

See Also

RshClient Class | SocketTools Namespace | RshClient.Login Overload List

RshClient.Login Method (String, Int32, String, String, Int32, RemoteOptions)

Establish an interactive terminal session for the specified user.

[Visual Basic]

```
Overloads Public Function Login( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userTerminal As String, _
   ByVal timeout As Integer, _
   ByVal options As RemoteOptions _
) As Boolean
```

[C#]

```
public bool Login(
   string hostName,
   int hostPort,
   string userName,
   string userTerminal,
   int timeout,
   RemoteOptions options
);
```

Parameters

hostName
A string which specifies the name of the server to connect to. The string may either be an IP address or a fully qualified domain name.

hostPort
An integer value which specifies the port number to connect to. The default port for this service is **rshPortLogin**.

userName
A string which specifies the username which used to authenticate the client session.

userTerminal
A string which specifies the client terminal type display purposes.

timeout
An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

options
One or more of the RemoteOptions enumeration flags.

Return Value

A boolean value which specifies if the connection has been established. If the socket is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the socket has been successfully created and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Login** method logs the specified user in on the remote host. Note that no password is provided to the remote host. This is because the remote login service uses ***user equivalence***. If the client system is recognized by the remote host as being equivalent, the login will proceed directly. If the client system is not recognized, the server will prompt the user for a password. For more information about user equivalence and the remote login service, refer to your server's operating system documentation.

Output from the command may be read using the **Read** method. Input can be supplied to the program using the **Write** method. To search for a specific sequence of bytes in the output returned by the server, use the **Search** method.

See Also

RshClient Class | SocketTools Namespace | RshClient.Login Overload List

RshClient.Login Method (String, String)

Establish an interactive terminal session for the specified user.

[Visual Basic]

```
Overloads Public Function Login( _
   ByVal hostName As String, _
   ByVal userName As String _
) As Boolean
```

[C#]

```
public bool Login(
   string hostName,
   string userName
);
```

Parameters

hostName

A string which specifies the name of the server to connect to. The string may either be an IP address or a fully qualified domain name.

userName

A string which specifies the username which used to authenticate the client session.

Return Value

A boolean value which specifies if the connection has been established. If the socket is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the socket has been successfully created and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Login** method logs the specified user in on the remote host. Note that no password is provided to the remote host. This is because the remote login service uses ***user equivalence***. If the client system is recognized by the remote host as being equivalent, the login will proceed directly. If the client system is not recognized, the server will prompt the user for a password. For more information about user equivalence and the remote login service, refer to your server's operating system documentation.

Output from the command may be read using the **Read** method. Input can be supplied to the program using the **Write** method. To search for a specific sequence of bytes in the output returned by the server, use the **Search** method.

See Also

RshClient Class | SocketTools Namespace | RshClient.Login Overload List

RshClient.Read Method

Read data from the server and store it in a byte array.

Overload List

Read data from the server and store it in a byte array.

public int Read(byte[]);

Read data from the server and store it in a byte array.

public int Read(byte[],int);

Read data from the server and store it in a string.

public int Read(ref string);

Read data from the server and store it in a string.

public int Read(ref string,int);

See Also

RshClient Class | SocketTools Namespace

RshClient.Read Method (Byte[])

Read data from the server and store it in a byte array.

[Visual Basic]
```
Overloads Public Function Read( _
   ByVal buffer As Byte() _
) As Integer
```

[C#]
```
public int Read(
   byte[] buffer
);
```

Parameters

buffer

A byte array that the data will be stored in.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the size of the byte array passed to the method. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

See Also

RshClient Class | SocketTools Namespace | RshClient.Read Overload List

RshClient.Read Method (Byte[], Int32)

Read data from the server and store it in a byte array.

[Visual Basic]

```
Overloads Public Function Read( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Read(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array that the data will be stored in.

length

An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the number of bytes specified. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

See Also

RshClient Class | SocketTools Namespace | RshClient.Read Overload List

RshClient.Read Method (String)

Read data from the server and store it in a string.

[Visual Basic]

```
Overloads Public Function Read( _
   ByRef buffer As String _
) As Integer
```

[C#]

```
public int Read(
   ref string buffer
);
```

Parameters

buffer

A string that will contain the data read from the client.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to a maximum of 4096 bytes. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

See Also

RshClient Class | SocketTools Namespace | RshClient.Read Overload List

RshClient.Read Method (String, Int32)

Read data from the server and store it in a string.

[Visual Basic]

```
Overloads Public Function Read( _
   ByRef buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Read(
   ref string buffer,
   int length
);
```

Parameters

buffer

A string that will contain the data read from the client.

length

An integer value which specifies the maximum number of bytes of data to read.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the number of bytes specified. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

See Also

RshClient Class | SocketTools Namespace | RshClient.Read Overload List

RshClient.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. If a socket has been allocated, it will be released and any active connections will be terminated. All properties will be reset to their default values.

See Also

RshClient Class | SocketTools Namespace

RshClient.Search Method

Search for a specific character sequence in the data stream.

Overload List

Search for a specific character sequence in the data stream.

public bool Search(string);

Search for a specific character sequence in the data stream.

public bool Search(string,byte[],ref int);

Search for a specific character sequence in the data stream.

public bool Search(string,ref string);

See Also

RshClient Class | SocketTools Namespace

RshClient.Search Method (String)

Search for a specific character sequence in the data stream.

[Visual Basic]

```
Overloads Public Function Search( _
   ByVal value As String _
) As Boolean
```

[C#]

```
public bool Search(
   string value
);
```

Parameters

value

A string argument which specifies the sequence of characters to search for in the data stream. When this sequence of characters is found, the method will return.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Search** method searches for a character sequence in the data stream and stops reading when it is found. This is useful when the client wants to automate responses to the server, such as executing a command and processing the output. The method will discard any data received up to and including the specified character sequence.

See Also

RshClient Class | SocketTools Namespace | RshClient.Search Overload List

RshClient.Search Method (String, Byte[], Int32)

Search for a specific character sequence in the data stream.

[Visual Basic]

```
Overloads Public Function Search( _
   ByVal value As String, _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool Search(
   string value,
   byte[] buffer,
   ref int length
);
```

Parameters

value

A string argument which specifies the sequence of characters to search for in the data stream. When this sequence of characters is found, the method will return.

buffer

An byte array that will contain the output sent by the server, up to and including the search string character sequence.

length

An integer value passed by reference which should be initialized to the maximum number of bytes of data to store in the buffer. When the method returns, this value will be updated with the actual number of bytes stored in the buffer.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Search** method searches for a character sequence in the data stream and stops reading when it is found. This is useful when the client wants to automate responses to the server, such as executing a command and processing the output. The method collects the output from the server and stores it in a buffer provided by the caller. When the method returns, the buffer will contain everything sent by the server up to and including the search string.

See Also

RshClient Class | SocketTools Namespace | RshClient.Search Overload List

RshClient.Search Method (String, String)

Search for a specific character sequence in the data stream.

```
[Visual Basic]
Overloads Public Function Search( _
   ByVal value As String, _
   ByRef buffer As String _
) As Boolean
```

```
[C#]
public bool Search(
   string value,
   ref string buffer
);
```

Parameters

value

A string argument which specifies the sequence of characters to search for in the data stream. When this sequence of characters is found, the method will return.

buffer

An string that will contain the output sent by the server, up to and including the search string character sequence.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Search** method searches for a character sequence in the data stream and stops reading when it is found. This is useful when the client wants to automate responses to the server, such as executing a command and processing the output. The method collects the output from the server and stores it in a buffer provided by the caller. When the method returns, the buffer will contain everything sent by the server up to and including the search string.

See Also

RshClient Class | SocketTools Namespace | RshClient.Search Overload List

RshClient.Uninitialize Method

Uninitialize the class library and release any resources allocated for the current thread.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method terminates any active connection, releases resources allocated for the current thread and unloads the networking library. After this method has been called, no further client operations may be performed until the class instance has been re-initialized.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the class is no longer needed.

See Also

RshClient Class | SocketTools Namespace | Initialize Method

RshClient.Write Method

Write one or more bytes of data to the server.

Overload List

Write one or more bytes of data to the server.

public int Write(byte[]);

Write one or more bytes of data to the server.

public int Write(byte[],int);

Write a character to the server.

public int Write(char);

Write one or more characters to the server.

public int Write(char,int);

Write a string of characters to the server.

public int Write(string);

Write a string of characters to the server.

public int Write(string,int);

See Also

RshClient Class | SocketTools Namespace

RshClient.Write Method (Byte[])

Write one or more bytes of data to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Write(
   byte[] buffer
);
```

Parameters

buffer

A byte array that contains the data to be written to the server.

Return Value

An integer value which specifies the number of bytes actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

RshClient Class | SocketTools Namespace | RshClient.Write Overload List

RshClient.Write Method (Byte[], Int32)

Write one or more bytes of data to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Write(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array that contains the data to be written to the server.

length

An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

RshClient Class | SocketTools Namespace | RshClient.Write Overload List

RshClient.Write Method (Char)

Write a character to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal key As Char _
) As Integer
```

[C#]

```
public int Write(
   char key
);
```

Parameters

key

A character which will be written to the server.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one character to the server. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

RshClient Class | SocketTools Namespace | RshClient.Write Overload List

RshClient.Write Method (Char, Int32)

Write one or more characters to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal key As Char, _
   ByVal repeat As Integer _
) As Integer
```

[C#]

```
public int Write(
   char key,
   int repeat
);
```

Parameters

key
A character which will be written to the server.

repeat
The number of characters that will be written to the server.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more characters to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

RshClient Class | SocketTools Namespace | RshClient.Write Overload List

RshClient.Write Method (String)

Write a string of characters to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String _
) As Integer
```

[C#]

```
public int Write(
   string buffer
);
```

Parameters

buffer

A string which contains the data to be written to the server.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

The string will be converted to an array of bytes before being written to the socket. By default, the character encoding used will be for the current locale. Depending on the contents of the string, the number of bytes written may be different than the string length specified. This is because the conversion from Unicode to a byte array may result in a multi-byte character sequence.

See Also

RshClient Class | SocketTools Namespace | RshClient.Write Overload List

RshClient.Write Method (String, Int32)

Write a string of characters to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Write(
   string buffer,
   int length
);
```

Parameters

buffer

A string which contains the data to be written to the server.

length

An integer value which specifies the maximum number of characters to write. This value cannot be larger than the length of the string specified by the caller.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

The string will be converted to an array of bytes before being written to the socket. By default, the character encoding used will be for the current locale. Depending on the contents of the string, the number of bytes written may be different than the string length specified. This is because the conversion from Unicode to a byte array may result in a multi-byte character sequence.

///

See Also

RshClient Class | SocketTools Namespace | RshClient.Write Overload List

RshClient Events

The events of the **RshClient** class are listed below. For a complete list of **RshClient** class members, see the RshClient Members topic.

Public Instance Events

⚡ OnCancel	Occurs when a blocking client operation is canceled.
⚡ OnConnect	Occurs when a connection is established with the remote host.
⚡ OnDisconnect	Occurs when the remote host disconnects from the local system.
⚡ OnError	Occurs when an client operation fails.
⚡ OnRead	Occurs when data is available to be read from the client.
⚡ OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
⚡ OnWrite	Occurs when data can be written to the client.

See Also

RshClient Class | SocketTools Namespace

RshClient.OnCancel Event

Occurs when a blocking client operation is canceled.

[Visual Basic]
```
Public Event OnCancel As EventHandler
```

[C#]
```
public event EventHandler OnCancel;
```

Remarks

The **OnCancel** event is generated when a blocking client operation, such as sending or receiving data, is canceled with the **Cancel** method. To assist in determining which operation was canceled, check the value of the **Status** property.

See Also

RshClient Class | SocketTools Namespace

RshClient.OnConnect Event

Occurs when a connection is established with the remote host.

[Visual Basic]

```
Public Event OnConnect As EventHandler
```

[C#]

```
public event EventHandler OnConnect;
```

Remarks

The **OnConnect** event occurs when a connection is made with a remote host as a result of a **Connect** method call. When the **Connect** method is called and the **Blocking** property is set to **false**, a socket is created but the connection is not actually established until after this event occurs. Between the time connection process is started and this event fires, no operation may be performed on the client other than calling the **Disconnect** method.

This event is only generated if the client is in non-blocking mode.

See Also

RshClient Class | SocketTools Namespace

RshClient.OnDisconnect Event

Occurs when the remote host disconnects from the local system.

[Visual Basic]
```
Public Event OnDisconnect As EventHandler
```

[C#]
```
public event EventHandler OnDisconnect;
```

Remarks

The **OnDisconnect** event occurs when the remote host closes its connection, terminating the client session with the application. Because there may still be data in the client receive buffers, you should continue to read data from the client until the **Read** method returns a value of 0. Once all of the data has been read, you should call the **Disconnect** method to close the local socket and release the resources allocated for the client.

This event is only generated if the client is in non-blocking mode.

See Also

RshClient Class | SocketTools Namespace

RshClient.OnError Event

Occurs when an client operation fails.

[Visual Basic]

```
Public Event OnError As OnErrorEventHandler
```

[C#]

```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type RshClient.ErrorEventArgs containing data related to this event. The following **RshClient.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a client operation fails.

See Also

RshClient Class | SocketTools Namespace

RshClient.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see RshClient.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.RshClient.ErrorEventArgs

[Visual Basic]
```
Public Class RshClient.ErrorEventArgs
    Inherits EventArgs
```

[C#]
```
public class RshClient.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.RshClient (in SocketTools.RshClient.dll)

See Also

RshClient.ErrorEventArgs Members | SocketTools Namespace

RshClient.ErrorEventArgs Members

RshClient.ErrorEventArgs overview

Public Instance Constructors

RshClient.ErrorEventArgs Constructor	Initializes a new instance of the RshClient.ErrorEventArgs class.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

RshClient.ErrorEventArgs Class | SocketTools Namespace

RshClient.ErrorEventArgs Constructor

Initializes a new instance of the RshClient.ErrorEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public RshClient.ErrorEventArgs();
```

See Also

RshClient.ErrorEventArgs Class | SocketTools Namespace

RshClient.ErrorEventArgs Properties

The properties of the **RshClient.ErrorEventArgs** class are listed below. For a complete list of **RshClient.ErrorEventArgs** class members, see the RshClient.ErrorEventArgs Members topic.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

RshClient.ErrorEventArgs Class | SocketTools Namespace

RshClient.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

RshClient.ErrorEventArgs Class | SocketTools Namespace | Error Property

RshClient.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public RshClient.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

RshClient.ErrorEventArgs Class | SocketTools Namespace | Description Property

RshClient.OnRead Event

Occurs when data is available to be read from the client.

[Visual Basic]

```
Public Event OnRead As EventHandler
```

[C#]

```
public event EventHandler OnRead;
```

Remarks

The **OnRead** event occurs when data is available to be read from the client. This event is level-triggered, which means that once this event fires, it will not occur again until some data has been read from the client. This design prevents an application from being flooded with event notifications. It is recommended that your application read all of the available data from the server and store it in a local buffer for processing. See the example below.

This event is only generated if the client is in non-blocking mode.

Example

```
Private Sub Socket_OnRead(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Socket.OnRead
    Dim strBuffer As String
    Dim nRead As Integer

    Do
        ' Read up to m_nBufferSize bytes of data from the server
        nRead = Socket.Read(strBuffer, m_nBufferSize)

        If nRead > 0 Then
            ' Append the data to an internal buffer for processing
            m_dataBuffer = m_dataBuffer + strBuffer
        End If
    Loop Until nRead < 1

    ProcessData()
End Sub
```

See Also

RshClient Class | SocketTools Namespace

RshClient.OnTimeout Event

Occurs when a blocking operation fails to complete before the timeout period elapses.

[Visual Basic]
```
Public Event OnTimeout As EventHandler
```

[C#]
```
public event EventHandler OnTimeout;
```

Remarks

The **OnTimeout** event occurs when a blocking operation, such as sending or receiving data on the client, fails to complete before the specified timeout period elapses. The timeout period for a blocking operation can be adjusted by setting the **Timeout** property.

This event is only generated if the client is in blocking mode.

See Also

RshClient Class | SocketTools Namespace

RshClient.OnWrite Event

Occurs when data can be written to the client.

[Visual Basic]

```
Public Event OnWrite As EventHandler
```

[C#]

```
public event EventHandler OnWrite;
```

Remarks

The **OnWrite** event occurs when the application can write data to the client. This event will typically occur when a connection is first established with the remote host, and after the **Write** method has failed because there was insufficient memory available in the client send buffers. In the second case, when some of the buffered data has been successfully sent to the remote host and there is space available in the send buffers, this event is used to signal the application that it may attempt to send more data.

This event is only generated if the client is in non-blocking mode.

See Also

RshClient Class | SocketTools Namespace

RshClient.ErrorCode Enumeration

Specifies the error codes returned by the RshClient class.

[Visual Basic]

```
Public Enum RshClient.ErrorCode
```

[C#]

```
public enum RshClient.ErrorCode
```

Remarks

The RshClient class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.
errorProxyAuthenticationRequired	Proxy authentication required.

errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.
errorUnsupportedMediaType	Request specified an unsupported media type.

errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.
errorNoMoreMailboxes	No more mailboxes exist on this server.

errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.
errorInvalidThreadHandle	The specified thread handle is no longer valid.

errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been defined.

errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or unrecognized.

errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.

Requirements

Namespace: SocketTools

Assembly: SocketTools.RshClient (in SocketTools.RshClient.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

RshClient.TraceOptions Enumeration

Specifies the logging options that the RshClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum RshClient.TraceOptions
```

[C#]

```
[Flags]
public enum RshClient.TraceOptions
```

Remarks

The RshClient class uses the **TraceOptions** enumeration to specify what kind of debugging information is written to the trace logfile. These options are only meaningful when trace logging is enabled by setting the **Trace** property to **true**.

Members

Member Name	Description	Value
traceDefault	The default trace logging option. This is the same as specifying the **traceInfo** option.	0
traceInfo	All network function calls are written to the trace file. This is the default value.	0
traceError	Only those network function calls which fail are recorded in the trace file.	1
traceWarning	Only those network function calls which fail, or return values which indicate a warning, are recorded in the trace file.	2
traceHexDump	All network function calls are written to the trace file, plus all the data that is sent or received is displayed, in both ASCII and hexadecimal format.	4
traceProcess	All function calls in the current process are logged, rather than only those functions in the current thread. This option is useful for multithreaded applications that are using worker threads.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.RshClient (in SocketTools.RshClient.dll)

See Also

SocketTools Namespace

RshClient.RemoteStatus Enumeration

Specifies the status values that may be returned by the RshClient class.

[Visual Basic]

```
Public Enum RshClient.RemoteStatus
```

[C#]

```
public enum RshClient.RemoteStatus
```

Remarks

The RshClient class uses the **RemoteStatus** enumeration to identify the current status of the client.

Members

Member Name	Description
statusUnused	A client session has not been created. Attempts to perform any network operations, such as sending or receiving data, will generate an error.
statusIdle	A client session has been created, but is not currently in use. A blocking socket operation can be executed at this point.
statusConnect	The client is in the process of establishing a connection with a remote host.
statusRead	The client is in the process of receiving data from a remote host.
statusWrite	The client is in the process of sending data to a remote host.
statusDisconnect	The client session is being closed and subsequent attempts to access the client will result in an error.

Requirements

Namespace: SocketTools

Assembly: SocketTools.RshClient (in SocketTools.RshClient.dll)

See Also

SocketTools Namespace

RshClient.RemoteOptions Enumeration

Specifies the options that the RshClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum RshClient.RemoteOptions
```

[C#]

```
[Flags]
public enum RshClient.RemoteOptions
```

Remarks

The RshClient class uses the **RemoteOptions** enumeration to specify one or more options to be used when establishing a connection with a remote host. Multiple options may be specified if necessary.

Members

Member Name	Description	Value
optionNone	No option specified.	0
optionDefault	The default connection option. This is the same as specifying **optionNone**.	0
optionReservedPort	This option specifies that a reserved port should be used to establish the connection. Reserved ports are those port numbers which are less than 1024. This option should be specified when connecting on the **rshPortShell** port.	1
optionFreeThread	This option specifies that class methods may be called from any thread, and not only the thread that established the connection. Using this option disables certain internal safety checks that are made by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads.	524288

Requirements

Namespace: SocketTools

Assembly: SocketTools.RshClient (in SocketTools.RshClient.dll)

See Also

SocketTools Namespace

RshClient.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub RshClient.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void RshClient.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ErrorEventArgs that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.RshClient (in SocketTools.RshClient.dll)

See Also

SocketTools Namespace

RshClient.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see RshClient.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.RshClient.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class RshClient.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class RshClient.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the RshClient class.

Example

```
<Assembly: SocketTools.RshClient.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.RshClient.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.RshClient (in SocketTools.RshClient.dll)

See Also

RshClient.RuntimeLicenseAttribute Members | SocketTools Namespace

RshClient.RuntimeLicenseAttribute Members

RshClient.RuntimeLicenseAttribute overview

Public Instance Constructors

RshClient.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

RshClient.RuntimeLicenseAttribute Class | SocketTools Namespace

RshClient.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public RshClient.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the RshClient class.

See Also

RshClient.RuntimeLicenseAttribute Class | SocketTools Namespace

RshClient.RuntimeLicenseAttribute Properties

The properties of the **RshClient.RuntimeLicenseAttribute** class are listed below. For a complete list of **RshClient.RuntimeLicenseAttribute** class members, see the RshClient.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

RshClient.RuntimeLicenseAttribute Class | SocketTools Namespace

RshClient.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

RshClient.RuntimeLicenseAttribute Class | SocketTools Namespace

RshClientException Class

The exception that is thrown when a client error occurs.

For a list of all members of this type, see RshClientException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.RshClientException

[Visual Basic]
```
Public Class RshClientException
    Inherits ApplicationException
```

[C#]
```
public class RshClientException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A RshClientException is thrown by the RshClient class when an error occurs.

The default constructor for the RshClientException class sets the **ErrorCode** property to the last client error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.RshClient (in SocketTools.RshClient.dll)

See Also

RshClientException Members | SocketTools Namespace

RshClientException Members

RshClientException overview

Public Instance Constructors

RshClientException	Overloaded. Initializes a new instance of the RshClientException class.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

RshClientException Class | SocketTools Namespace

RshClientException Constructor

Initializes a new instance of the RshClientException class with the last network error code.

Overload List

Initializes a new instance of the RshClientException class with the last network error code.

public RshClientException();

Initializes a new instance of the RshClientException class with a specified error number.

public RshClientException(int);

Initializes a new instance of the RshClientException class with a specified error message.

public RshClientException(string);

Initializes a new instance of the RshClientException class with a specified error message and a reference to the inner exception that is the cause of this exception.

public RshClientException(string,Exception);

See Also

RshClientException Class | SocketTools Namespace

RshClientException Constructor ()

Initializes a new instance of the RshClientException class with the last network error code.

[Visual Basic]

```
Overloads Public Sub New()
```

[C#]

```
public RshClientException();
```

Remarks

The ctor constructor sets the **ErrorCode** property to the last client error that occurred. For more information about the errors that may occur, refer to the RshClient.ErrorCode enumeration.

See Also

RshClientException Class | SocketTools Namespace | RshClientException Constructor Overload List

RshClientException Constructor (String)

Initializes a new instance of the RshClientException class with a specified error message.

[Visual Basic]
```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]
```
public RshClientException(
   string message
);
```

Parameters

message
The error message that explains the reason for the exception.

Remarks

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

RshClientException Class | SocketTools Namespace | RshClientException Constructor Overload List

RshClientException Constructor (String, Exception)

Initializes a new instance of the RshClientException class with a specified error message and a reference to the inner exception that is the cause of this exception.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal innerException As Exception _
)
```

[C#]

```
public RshClientException(
   string message,
   Exception innerException
);
```

Parameters

message

The error message that explains the reason for the exception.

innerException

The exception that is the cause of the current exception. If the ***innerException*** parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

RshClientException Class | SocketTools Namespace | RshClientException Constructor Overload List

RshClientException Constructor (Int32)

Initializes a new instance of the RshClientException class with a specified error number.

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

```
[C#]
public RshClientException(
   int code
);
```

Parameters

code

An integer value which specifies an error code.

Remarks

This constructor sets the **ErrorCode** property to the specified error code. For more information about the errors that may occur, refer to the RshClient.ErrorCode enumeration.

See Also

RshClientException Class | SocketTools Namespace | RshClientException Constructor Overload List

RshClientException Properties

The properties of the **RshClientException** class are listed below. For a complete list of **RshClientException** class members, see the RshClientException Members topic.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

RshClientException Class | SocketTools Namespace

RshClientException.ErrorCode Property

Gets a value which specifies the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property ErrorCode As ErrorCode
```

[C#]

```
public RshClient.ErrorCode ErrorCode {get;}
```

Property Value

Returns a RshClient.ErrorCode enumeration value which specifies the error code.

Remarks

The **ErrorCode** property returns the error code that specifies the cause of the exception.

The default constructor for the RshClientException class sets the error code to the last network error that occurred. For more information about the errors that may occur, refer to the RshClient.ErrorCode enumeration.

See Also

RshClientException Class | SocketTools Namespace

RshClientException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]

```
Overrides Public ReadOnly Property Message As String
```

[C#]

```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

RshClientException Class | SocketTools Namespace

RshClientException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception. This value is the same as the values returned by the Windows Sockets API. For more information about socket error codes, see the Windows Socket Version 2 API error code documentation in MSDN.

See Also

RshClientException Class | SocketTools Namespace

RshClientException Methods

The methods of the **RshClientException** class are listed below. For a complete list of **RshClientException** class members, see the RshClientException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

RshClientException Class | SocketTools Namespace

RshClientException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]
```
Overrides Public Function ToString() As String
```

[C#]
```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

RshClientException Class | SocketTools Namespace

SmtpClient Class

Implements the Simple Mail Transfer Protocol.

For a list of all members of this type, see SmtpClient Members.

System.Object
  SocketTools.SmtpClient

[Visual Basic]
```
Public Class SmtpClient
    Implements IDisposable
```

[C#]
```
public class SmtpClient : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The Simple Mail Transfer Protocol (SMTP) enables applications to deliver email messages to one or more recipients. The class provides an interface for addressing and delivering messages, and extended features such as user authentication and delivery status notification. There is no requirement to have a specific email application installed or certain types of servers installed on the local system. The class can be used to deliver mail through a wide variety of systems, from standard UNIX based mail servers to Windows systems running Exchange or Lotus Notes and Domino.

Using this class, messages can be delivered directly to the recipient, or they can be routed through a relay server, such as an Internet Service Provider's mail system. The SocketTools.MailMessage class can be integrated with this class in order to provide an extremely simple, yet flexible interface for composing and delivering messages.

This class supports secure connections using the standard TLS protocols. Both implicit and explicit TLS connections can be established, enabling the class to work with a wide variety of servers.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SmtpClient (in SocketTools.SmtpClient.dll)

See Also

SmtpClient Members | SocketTools Namespace

SmtpClient Members

SmtpClient overview

Public Static (Shared) Fields

smtpPortDefault	A constant value which specifies the default port number.
smtpPortSecure	A constant value which specifies the default port number for a secure connection.
smtpTimeout	A constant value which specifies the default timeout period.

Public Instance Constructors

SmtpClient Constructor	Initializes a new instance of the SmtpClient class.

Public Instance Fields

Recipient	Gets and sets the recipients specified for the current message.

Public Instance Properties

Authentication	Gets and sets the method used to authenticate the user.
AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
BearerToken	Gets and sets the bearer token used with OAuth 2.0 authentication.
Blocking	Gets and sets a value which indicates if the client is in blocking mode.
CertificateExpires	Get a value that specifies the date that the security certificate expires.
CertificateIssued	Get a value that specifies the date that the security certificate was issued.
CertificateIssuer	Get a value that provides information about the organization that issued the certificate.
CertificateName	Gets and sets a value that specifies the name of the client certificate.
CertificatePassword	Gets and sets the password associated with the client certificate.
CertificateStatus	Gets a value which indicates the status of the security certificate returned by the remote host.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateSubject	Gets a value that provides information about the

	organization that the server certificate was issued to.
CertificateUser	Gets and sets the user that owns the client certificate.
CipherStrength	Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.
Extended	Gets and sets a value that specifies if extended SMTP options should be enabled.
Handle	Gets a value that specifies the client handle allocated for the current session.
HashStrength	Gets a value which specifies the length of the message digest that was selected for a secure connection.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsConnected	Gets a value which indicates if a connection to the remote host has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsReadable	Gets a value which indicates if there is data available to be read from the socket connection to the server.
IsWritable	Gets a value which indicates if data can be written to the client without blocking.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets the local Internet address that the client is bound to.
LocalDomain	Gets and sets the local domain name.
Localize	Gets a value that specifies if the date and time are localized.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
Options	Gets and sets a value which specifies one or more client options.

Password	Gets and sets the password used to authenticate the client.
Recipients	Gets the number of current message recipients.
RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.
ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.
ReturnReceipt	Enables and disables delivery status notification.
Secure	Gets and sets a value which specifies if a secure connection is established.
SecureCipher	Gets a value that specifies the encryption algorithm used for a secure connection.
SecureHash	Gets a value that specifies the message digest algorithm used for a secure connection.
SecureKeyExchange	Gets a value that specifies the key exchange algorithm used for a secure connection.
SecureProtocol	Gets and sets a value which specifies the protocol used for a secure connection.
Sender	Gets and sets the sender email address.
Status	Gets a value which specifies the current status of the client.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
TimeZone	Gets and sets the current timezone offset in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
UserName	Gets and sets the username used to authenticate the client session.

Version	Gets a value which returns the current version of the SmtpClient class library.

Public Instance Methods

AddRecipient	Add an address to the recipient list for the current message.
AppendMessage	Overloaded. Append text to the current message being composed.
AttachThread	Attach an instance of the class to the current thread
Authenticate	Overloaded. Authenticate the client session with a username and password.
Cancel	Cancel the current blocking client operation.
CloseMessage	Closes the current message.
Command	Overloaded. Send a custom command to the server.
Connect	Overloaded. Establish a connection with a remote host.
CreateMessage	Overloaded. Begin the composition of a new message to be delivered.
Disconnect	Terminate the connection with a remote host.
Dispose	Overloaded. Releases all resources used by SmtpClient.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
ExpandAddress	Expand the specified email address.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the SmtpClient class.
Reset	Reset the internal state of the object, resetting all properties to their default values.
SendMessage	Overloaded. Submit the specified message to the mail server for delivery.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
VerifyAddress	Verify the specified email address.
Write	Overloaded. Write one or more bytes of data to the server.

Public Instance Events

OnCancel	Occurs when a blocking client operation is canceled.
OnCommand	Occurs when the client sends a command to the remote host and receives a reply indicating the result of that command.
OnConnect	Occurs when a connection is established with the remote host.
OnDisconnect	Occurs when the remote host disconnects from the local system.
OnError	Occurs when an client operation fails.
OnProgress	Occurs as a data stream is being read or written to the client.
OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the SmtpClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient Constructor

Initializes a new instance of the SmtpClient class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public SmtpClient();
```

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient Fields

The fields of the **SmtpClient** class are listed below. For a complete list of **SmtpClient** class members, see the SmtpClient Members topic.

Public Static (Shared) Fields

smtpPortDefault	A constant value which specifies the default port number.
smtpPortSecure	A constant value which specifies the default port number for a secure connection.
smtpTimeout	A constant value which specifies the default timeout period.

Public Instance Fields

Recipient	Gets and sets the recipients specified for the current message.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.Recipient Field

Gets and sets the recipients specified for the current message.

[Visual Basic]
```
Public ReadOnly Recipient As RecipientArray
```

[C#]
```
public readonly RecipientArray Recipient;
```

Remarks

The **Recipient** array is used to enumerate the recipient addresses that have been specified for the current message. The list of message recipients managed using this property is used by the **SendMessage** method when delivering the message. This array is zero based, meaning that the first index value is zero. The total number of recipients specified in the message can be determined by checking the value of the **Recipients** property.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient Properties

The properties of the **SmtpClient** class are listed below. For a complete list of **SmtpClient** class members, see the SmtpClient Members topic.

Public Instance Properties

Authentication	Gets and sets the method used to authenticate the user.
AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
BearerToken	Gets and sets the bearer token used with OAuth 2.0 authentication.
Blocking	Gets and sets a value which indicates if the client is in blocking mode.
CertificateExpires	Get a value that specifies the date that the security certificate expires.
CertificateIssued	Get a value that specifies the date that the security certificate was issued.
CertificateIssuer	Get a value that provides information about the organization that issued the certificate.
CertificateName	Gets and sets a value that specifies the name of the client certificate.
CertificatePassword	Gets and sets the password associated with the client certificate.
CertificateStatus	Gets a value which indicates the status of the security certificate returned by the remote host.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateSubject	Gets a value that provides information about the organization that the server certificate was issued to.
CertificateUser	Gets and sets the user that owns the client certificate.
CipherStrength	Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.
Extended	Gets and sets a value that specifies if extended SMTP options should be enabled.
Handle	Gets a value that specifies the client handle allocated for the current session.
HashStrength	Gets a value which specifies the length of the message digest that was selected for a secure connection.

HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsConnected	Gets a value which indicates if a connection to the remote host has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsReadable	Gets a value which indicates if there is data available to be read from the socket connection to the server.
IsWritable	Gets a value which indicates if data can be written to the client without blocking.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets the local Internet address that the client is bound to.
LocalDomain	Gets and sets the local domain name.
Localize	Gets a value that specifies if the date and time are localized.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
Options	Gets and sets a value which specifies one or more client options.
Password	Gets and sets the password used to authenticate the client.
Recipients	Gets the number of current message recipients.
RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.
ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.
ReturnReceipt	Enables and disables delivery status notification.
Secure	Gets and sets a value which specifies if a secure

	connection is established.
SecureCipher	Gets a value that specifies the encryption algorithm used for a secure connection.
SecureHash	Gets a value that specifies the message digest algorithm used for a secure connection.
SecureKeyExchange	Gets a value that specifies the key exchange algorithm used for a secure connection.
SecureProtocol	Gets and sets a value which specifies the protocol used for a secure connection.
Sender	Gets and sets the sender email address.
Status	Gets a value which specifies the current status of the client.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
TimeZone	Gets and sets the current timezone offset in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
UserName	Gets and sets the username used to authenticate the client session.
Version	Gets a value which returns the current version of the SmtpClient class library.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.Authentication Property

Gets and sets the method used to authenticate the user.

[Visual Basic]

```
Public Property Authentication As SmtpAuthentication
```

[C#]

```
public SmtpClient.SmtpAuthentication Authentication {get; set;}
```

Property Value

A SmtpAuthentication enumeration value which specifies the authentication method.

Remarks

The **authXOAuth2** and **authBearer** authentication methods are similar, but they are not interchangeable. Both use an OAuth 2.0 bearer token to authenticate the client session, but they differ in how the token is presented to the server. It is currently preferable to use the XOAUTH2 method because it is more widely available and some service providers do not yet support the OAUTHBEARER method.

See Also

SmtpClient Class | SocketTools Namespace | BearerToken Poperty | Password Property | UserName Property

SmtpClient.AutoResolve Property

Gets and sets a value that determines if host names and addresses are automatically resolved.

[Visual Basic]

```
Public Property AutoResolve As Boolean
```

[C#]

```
public bool AutoResolve {get; set;}
```

Property Value

Returns **true** if host names are automatically resolved to Internet addresses. The default value is **false**.

Remarks

Setting the **AutoResolve** property determines if the class automatically resolves host names and addresses specified by the **HostName** and **HostAddress** properties. If set to **true**, setting the **HostName** property will cause the class to automatically determine the corresponding IP address and update the **HostAddress** property accordingly. Likewise, setting the **HostAddress** property will cause the class to determine the host name and update the **HostName** property. Setting this property to **false** prevents the class from resolving host names until a connection attempt is made.

It is important to note that setting the **HostName** or **HostAddress** property may cause the current thread to block, sometimes for several seconds, until the name or address is resolved. To prevent this behavior, set this property value to **false**.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.BearerToken Property

Gets and sets the bearer token used with OAuth 2.0 authentication.

[Visual Basic]
```
Public Property BearerToken As String
```

[C#]
```
public string BearerToken {get; set;}
```

Property Value

Returns a string which contains the bearer token. Assigning a value to this property sets the curent authentication type to use OAuth 2.0 authentication and updates the bearer token.

Remarks

Assigning a value to the **BearerToken** property will automatically change the current authentication method to use OAuth 2.0 if necessary.

You should only use an OAuth 2.0 authentication method if you understand the process of how to request the access token. Obtaining a bearer token requires registering your application with the mail service provider (e.g.: Microsoft or Google), getting a unique client ID associated with your application and then requesting the bearer token using the appropriate scope for the service. Obtaining the initial token will typically involve interactive confirmation on the part of the user, requiring they grant permission to your application to access their mail account.

Your application should not store the bearer token for later use. They usually have a relatively short lifespan, typically about an hour, and are designed to be used with the current client session. You should specify offline access as part of the OAuth 2.0 scope, and store the refresh token provided by the service. The refresh token has a much onger validity period and can be used to obtain a new access token when needed.

If the current authentication method does not use OAuth 2.0, this property will return an empty string and you should use the **Password** property to obtain the current user password.

See Also

SmtpClient Class | SocketTools Namespace | Authentication Property | Password Property | UserName Property

SmtpClient.Blocking Property

Gets and sets a value which indicates if the client is in blocking mode.

[Visual Basic]

```
Public Property Blocking As Boolean
```

[C#]

```
public bool Blocking {get; set;}
```

Property Value

Returns **true** if the client is in blocking mode; otherwise it returns **false**. The default value is **true**.

Remarks

Setting the **Blocking** property determines if client operations complete synchronously or asynchronously. If set to **true**, then each client operation (such as sending or receiving data) will return when the operation has completed or timed-out. If set to **false**, client operations will return immediately. If the operation would result in the client blocking (such as attempting to read data when no data has been sent by the remote host), an error is generated.

It is important to note that certain events, such as **OnConnect OnDisconnect** are only fired if the client is in non-blocking mode.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.CertificateExpires Property

Get a value that specifies the date that the security certificate expires.

[Visual Basic]

```
Public ReadOnly Property CertificateExpires As String
```

[C#]

```
public string CertificateExpires {get;}
```

Property Value

A string which specifies a date using the local date and time format.

Remarks

The **CertificateExpires** property returns a string that specifies the date and time that the security certificate expires. This property will return an empty string if a secure connection has not been established with the remote host.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.CertificateIssued Property

Get a value that specifies the date that the security certificate was issued.

[Visual Basic]

```
Public ReadOnly Property CertificateIssued As String
```

[C#]

```
public string CertificateIssued {get;}
```

Property Value

A string which specifies a date using the local date and time format.

Remarks

The **CertificateIssued** property returns a string that specifies the date and time that the security certificate was issued. This property will return an empty string if a secure connection has not been established with the remote host.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.CertificateIssuer Property

Get a value that provides information about the organization that issued the certificate.

[Visual Basic]

```
Public ReadOnly Property CertificateIssuer As String
```

[C#]

```
public string CertificateIssuer {get;}
```

Property Value

A string that contains a comma separated list of name value pairs.

Remarks

The **CertificateIssuer** property returns a string that contains information about the organization that issued the server certificate. The string value is a comma separated list of tagged name and value pairs. In the nomenclature of the X.500 standard, each of these pairs are called a relative distinguished name (RDN), and when concatenated together, forms the issuer's distinguished name (DN). For example:

```
C=US, O="RSA Data Security, Inc.", OU=Secure Server Certification Authority
```

To obtain a specific value, such as the name of the issuer or the issuer's country, the application must parse the string returned by this property. Some of the common tokens used in the distinguished name are:

Token	Description
C	The ISO standard two character country code.
S	The name of the state or province.
L	The name of the city or locality.
O	The name of the company or organization.
OU	The name of the department or organizational unit
CN	The common name; with X.509 certificates, this is the domain name of the site the certificate was issued for.

This property will return an empty string if a secure connection has not been established with the remote host.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.CertificateName Property

Gets and sets a value that specifies the name of the client certificate.

[Visual Basic]

```
Public Property CertificateName As String
```

[C#]

```
public string CertificateName {get; set;}
```

Property Value

A string which specifies the certificate name.

Remarks

The **CertificateName** property is used to specify the name of a client certificate to use when establishing a secure connection. It is only required that you set this property value if the server requires a client certificate for authentication. If this property is not set, a client certificate will not be provided to the server. If a certificate name is specified, the certificate must have a private key associated with it, otherwise the connection attempt will fail because the control will be unable to create a security context for the session.

When the certificate store is searched for a matching certificate, it will first search for any certificate with a friendly name that matches the property value. If no valid certificate is found, it will then search for a certificate with a matching common name.

Certificates may be installed and viewed on the local system using the Certificate Manager that is included with the Windows operating system. For more information, refer to the documentation for the Microsoft Management Console.

See Also

SmtpClient Class | SocketTools Namespace | CertificateStore Property | Secure Property

SmtpClient.CertificateStatus Property

Gets a value which indicates the status of the security certificate returned by the remote host.

[Visual Basic]

```
Public ReadOnly Property CertificateStatus As SecurityCertificate
```

[C#]

```
public SmtpClient.SecurityCertificate CertificateStatus {get;}
```

Property Value

A SecurityCertificate enumeration value which specifies the status of the certificate.

Remarks

The **CertificateStatus** property is used to determine the status of the security certificate returned by the remote host when a secure connection has been established. This property value should be checked after the connection to the server has completed, but prior to beginning a transaction.

Note that if the certificate cannot be validated, the secure connection will not be automatically terminated. It is the responsibility of your application to determine the best course of action to take if the certificate is invalid. Even if the security certificate cannot be validated, the data exchanged with the remote host will still be encrypted.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.CertificateStore Property

Gets and sets a value that specifies the name of the local certificate store.

[Visual Basic]

```
Public Property CertificateStore As String
```

[C#]

```
public string CertificateStore {get; set;}
```

Property Value

A string which specifies the certificate store name. The default value is the current user's personal certificate store.

Remarks

The **CertificateStore** property is used to specify the name of the certificate store which contains the security certificate to use when establishing a secure connection. The certificate may either be stored in the registry or in a file. If the certificate is stored in the registry, then this property should be set to one of the following predefined values:

Store Name	Description
CA	Certification authority certificates. These are certificates that are issued by entities which are entrusted to issue certificates to other individuals or organizations. Companies such as VeriSign and Thawte act as certification authorities.
MY	Personal certificates and their associated private keys for the current user. This store typically holds the client certificates used to establish a user's credentials. If a certificate store is not specified, this is the default value that is used.
ROOT	Certificates that have been self-signed by a certificate authority. Root certificates for a number of different certification authorities such as VeriSign and Thawte are installed as part of the operating system and periodically updated by Microsoft.

In most cases the client certificate will be installed in the user's personal certificate store, and therefore it is not necessary to set this property value because that is the default location that will be used to search for the certificate. This property is only used if the **CertificateName** property is also set to a valid certificate name.

If you are using a local certificate store, with the certificate and private key stored in the registry, you can explicitly specify whether the certificate store for the current user or the local machine (all users) should be used. This is done by prefixing the certificate store name with "HKCU" for the current user, or "HKLM" for the local machine. For example, a certificate store name of "HKLM:MY" would specify the personal certificate store for the local machine, rather than the current user. If neither prefix is specified, it will default to the certificate store for the current user.

This property may also be used to specify a file that contains the client certificate. In this case, the property should specify the full path to the file and must contain both the certificate and private key in PKCS12

format. If the file is protected by a password, the **CertificatePassword** property must also be set to specify the password.

See Also

SmtpClient Class | SocketTools Namespace | CertificatePassword Property | Secure Property

SmtpClient.CertificateSubject Property

Gets a value that provides information about the organization that the server certificate was issued to.

[Visual Basic]
```
Public ReadOnly Property CertificateSubject As String
```

[C#]
```
public string CertificateSubject {get;}
```

Property Value

A string that contains a comma separated list of name value pairs.

Remarks

The **CertificateSubject** property returns a string that contains information about the organization that the server certificate was issued to. The string value is a comma separated list of tagged name and value pairs. In the nomenclature of the X.500 standard, each of these pairs are called a relative distinguished name (RDN), and when concatenated together, forms the issuer's distinguished name (DN). For example:

```
C=US, O="RSA Data Security, Inc.", OU=Secure Server Certification Authority
```

To obtain a specific value, such as the name of the issuer or the issuer's country, the application must parse the string returned by this property. Some of the common tokens used in the distinguished name are:

Token	Description
C	The ISO standard two character country code.
S	The name of the state or province.
L	The name of the city or locality.
O	The name of the company or organization.
OU	The name of the department or organizational unit
CN	The common name; with X.509 certificates, this is the domain name of the site the certificate was issued for.

This property will return an empty string if a secure connection has not been established with the remote host.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.CipherStrength Property

Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.

[Visual Basic]

```
Public ReadOnly Property CipherStrength As Integer
```

[C#]

```
public int CipherStrength {get;}
```

Property Value

An integer value which specifies the encryption key length if a secure connection has been established; otherwise a value of 0 is returned.

Remarks

The **CipherStrength** property returns the number of bits in the key used to encrypt the secure data stream. Common values returned by this property are 128 and 256. A key length of 40 or 56 bits is considered insecure and subject to brute force attacks. 128-bit and 256-bit keys are considered secure. If this property returns a value of 0, this means that a secure connection has not been established with the remote host.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.Handle Property

Gets a value that specifies the client handle allocated for the current session.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer which represents a client handle. If there is no active connection, a value of -1 is returned.

Remarks

The **Handle** property specifies the numeric descriptor of the current client session.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.HashStrength Property

Gets a value which specifies the length of the message digest that was selected for a secure connection.

[Visual Basic]

```
Public ReadOnly Property HashStrength As Integer
```

[C#]

```
public int HashStrength {get;}
```

Property Value

An integer value which specifies the length of the message digest if a secure connection has been established; otherwise a value of 0 is returned.

Remarks

The **HashStrength** property returns the number of bits used in the message digest (hash) that was selected. Common values returned by this property are 128 and 160. If this property returns a value of 0, this means that a secure connection has not been established with the remote host.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.HostAddress Property

Gets and sets a value which specifies the Internet address used to establish a connection.

[Visual Basic]
```
Public Property HostAddress As String
```

[C#]
```
public string HostAddress {get; set;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **HostAddress** property can be used to set the Internet address for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the address is assigned to a valid host name, the **HostName** property will be updated with that value.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.HostName Property

Gets and sets a value which specifies the host name used to establish a connection.

[Visual Basic]
```
Public Property HostName As String
```

[C#]
```
public string HostName {get; set;}
```

Property Value

A string which specifies a host name.

Remarks

The **HostName** property can be used to set the host name for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the name can be resolved to a valid Internet address, the **HostAddress** property will be updated with that value.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.IsBlocked Property

Gets a value which indicates if the current thread is performing a blocking client operation.

[Visual Basic]

```
Public ReadOnly Property IsBlocked As Boolean
```

[C#]

```
public bool IsBlocked {get;}
```

Property Value

Returns **true** if the current thread is blocking, otherwise returns **false**.

Remarks

The **IsBlocked** property returns **true** if the current thread is blocked performing an operation. Because the Windows Sockets API only permits one blocking operation per thread of execution, this property should be checked before starting any blocking operation in response to an event.

If the **IsBlocked** property returns **false**, this means there are no blocking operations on the current thread at that time. However, this does not guarantee that the next client operation will not fail. An application should always check the return value from a client operation and check the value of the **LastError** property if an error occurs.

Note that this property will return **true** if there is any blocking operation being performed by the current thread, regardless of whether this specific instance of the class is responsible for the blocking operation or not.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.IsConnected Property

Gets a value which indicates if a connection to the remote host has been established.

[Visual Basic]

```
Public ReadOnly Property IsConnected As Boolean
```

[C#]

```
public bool IsConnected {get;}
```

Property Value

Returns **true** if the connection has been established; otherwise returns **false**.

Remarks

The **IsConnected** property can only be used to indicate if there is still a logical connection to the remote host. It cannot be used to detect abnormal conditions such as the remote host aborting the connection, the physical network connection being lost or other critical errors.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.IsInitialized Property

Gets a value which indicates if the current instance of the class has been initialized successfully.

[Visual Basic]
```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]
```
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the class instance has been initialized; otherwise returns **false**.

Remarks

The **IsInitialized** property is used to determine if the current instance of the class has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the class may not be able to initialize itself.

The most common reasons that a class instance may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required networking libraries or not being able to access the system registry.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.IsReadable Property

Gets a value which indicates if there is data available to be read from the socket connection to the server.

[Visual Basic]

```
Public ReadOnly Property IsReadable As Boolean
```

[C#]

```
public bool IsReadable {get;}
```

Property Value

Returns **true** if there is data available to be read; otherwise returns **false**.

Remarks

The **IsReadable** property returns **true** if data can be read from the client without blocking. For non-blocking sessions, this property can be checked before the application attempts to read the client. Note that even if this property does return **true** indicating that there is data available to be read, applications should always check the return value from the **Read** method.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.IsWritable Property

Gets a value which indicates if data can be written to the client without blocking.

[Visual Basic]

```
Public ReadOnly Property IsWritable As Boolean
```

[C#]

```
public bool IsWritable {get;}
```

Property Value

Returns **true** if data can be written to the client; otherwise returns **false**.

Remarks

The **IsWritable** property returns **true** if data can be written to the client without blocking. For non-blocking sessions, this property can be checked before the application attempts to write data to the client. Note that even if this property does return **true** indicating that data can be written to the client, applications should always check the return value from the **Write** method.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public SmtpClient.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.LocalAddress Property

Gets the local Internet address that the client is bound to.

[Visual Basic]

```
Public ReadOnly Property LocalAddress As String
```

[C#]

```
public string LocalAddress {get;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **LocalAddress** property returns the local Internet address that the client is bound to when a connection is established with a remote host. This property may return either an IPv4 or IPv6 formatted address, depending on the type of connection that was established.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.LocalDomain Property

Gets and sets the local domain name.

[Visual Basic]

```
Public Property LocalDomain As String
```

[C#]

```
public string LocalDomain {get; set;}
```

Property Value

A string which specifies the local domain name.

Remarks

The **LocalDomain** returns the local domain name used when the client identifies itself to the mail server. If this property is an empty string, then the control will attempt to automatically determine the appropriate domain name to use based on the system configuration. Setting this property will cause the control to use that value when identifying itself to the server.

This property should only be set if it is absolutely necessary. In most cases, it is preferable to leave this property undefined and allow the control to automatically determine the correct domain name to use. Setting an invalid domain name may cause the mail server to reject the connection.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.Localize Property

Gets a value that specifies if the date and time are localized.

[Visual Basic]

```
Public Property Localize As Boolean
```

[C#]

```
public bool Localize {get; set;}
```

Property Value

A boolean value which specifies if the date and time is localized.

Remarks

Setting the **Localize** property controls how date and time values are localized. If the property is set to **true**, then the date and time will be adjusted to the current timezone. If the property is set to **false**, the date and time are specified as UTC (Coordinated Universal Time) values.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.LocalName Property

Gets a value which specifies the host name for the local system.

[Visual Basic]

```
Public ReadOnly Property LocalName As String
```

[C#]

```
public string LocalName {get;}
```

Property Value

A string which specifies the local host name.

Remarks

The **LocalName** property returns the fully-qualified host name assigned to the local system. If the system has not been configured with an Internet domain name, then this property will return the NetBIOS name assigned to the local system.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.LocalPort Property

Gets the local port number the client is bound to.

[Visual Basic]

```
Public ReadOnly Property LocalPort As Integer
```

[C#]

```
public int LocalPort {get;}
```

Property Value

An integer value which specifies a port number. The default value is 0.

Remarks

The **LocalPort** property is used to identify the local port number that the client is bound to to when a connection is established with a remote host.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.Options Property

Gets and sets a value which specifies one or more client options.

[Visual Basic]

```
Public Property Options As SmtpOptions
```

[C#]

```
public SmtpClient.SmtpOptions Options {get; set;}
```

Property Value

Returns one or more SmtpOptions enumeration flags which specify the options for the client. The default value for this property is **smtpOptionNone**.

Remarks

The **Options** property specifies one or more default options options which are used when establishing a connection using the **Connect** method.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.Password Property

Gets and sets the password used to authenticate the client.

[Visual Basic]

```
Public Property Password As String
```

[C#]

```
public string Password {get; set;}
```

Property Value

A string which specifies the password.

Remarks

The **Password** property specifies the password used to authenticate the client session. This property is used as the default value for the **Authenticate** method if no password is specified as an argument.

Refer to the **Authentication** property for more information on the available authentication methods. If you are using the OAuth 2.0 authentication method, this property should not be set to the user's password. Instead, you should set the **BearerToken** property to the OAuth 2.0 bearer token issued by the mail service provider. Note that these access tokens can be much larger than your typical password and are only valid for a limited period of time.

You can use the **Password** property to specify an OAuth 2.0 bearer token. However, it is recommended that you use the **BearerToken** property instead of assigning it to this property. It will ensure compatibility with future versions of the class and make it clear in your code you are using an OAuth 2.0 bearer token and not a password. If the **Authentication** property specifies one of the OAuth 2.0 authentication methods, this property will return the bearer token.

See Also

SmtpClient Class | SocketTools Namespace | Authentication Property | BearerToken Property | UserName Property | Authenticate Method

SmtpClient.Recipients Property

Gets the number of current message recipients.

[Visual Basic]

```
Public ReadOnly Property Recipients As Integer
```

[C#]

```
public int Recipients {get;}
```

Property Value

An integer which specifies the number of recipients.

Remarks

The **Recipients** property specifies the number of recipient addresses which have been specified using the **Recipient** property array. The maximum number of recipients for a message varies by server, but is typically limited to approximately 100 addresses.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.RemotePort Property

Gets and sets a value which specifies the remote port number.

[Visual Basic]

```
Public Property RemotePort As Integer
```

[C#]

```
public int RemotePort {get; set;}
```

Property Value

An integer value which specifies a port number.

Remarks

The **RemotePort** property is used to set the port number that will be used to establish a connection with a remote host. If the port number specifies a well-known port, the **RemoteService** property will be updated with that name.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.ResultCode Property

Gets a value which specifies the last result code returned by the server.

[Visual Basic]
```
Public ReadOnly Property ResultCode As Integer
```

[C#]
```
public int ResultCode {get;}
```

Property Value

An integer value which specifies the last result code returned by the server.

Remarks

Result codes are three-digit numeric values returned by the remote server and may be broken down into the following ranges:

ResultCode	Description
100-199	Positive preliminary result. This indicates that the requested action is being initiated, and the client should expect another reply from the server before proceeding.
200-299	Positive completion result. This indicates that the server has successfully completed the requested action.
300-399	Positive intermediate result. This indicates that the requested action cannot complete until additional information is provided to the server.
400-499	Transient negative completion result. This indicates that the requested action did not take place, but the error condition is temporary and may be attempted again.
500-599	Permanent negative completion result. This indicates that the requested action did not take place.

It is important to note that while some result codes have become standardized, not all servers respond to commands using the same result codes. For example, one server may respond with a result code of 221 to indicate success, while another may respond with a value of 235. It is recommended that applications check for ranges of values to determine if a command was successful, not a specific value.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.ResultString Property

Gets a string value which describes the result of the previous command.

[Visual Basic]
```
Public ReadOnly Property ResultString As String
```

[C#]
```
public string ResultString {get;}
```

Property Value

A string which describes the result of the previous command executed on the server.

Remarks

The **ResultString** property returns the result string from the last action taken by the client. This string is generated by the remote server, and typically is used to describe the result code. For example, if an error is indicated by the result code, the result string may describe the condition that caused the error.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.ReturnReceipt Property

Enables and disables delivery status notification.

[Visual Basic]

```
Public Property ReturnReceipt As Boolean
```

[C#]

```
public bool ReturnReceipt {get; set;}
```

Property Value

A boolean value which specifies if delivery status notification has been enabled.

Remarks

The **ReturnReceipt** property enables or disables delivery status notification (DSN) by the mail server. If the property is set to True, a mail message will be automatically returned to the sender indicating if the message was delivered successfully, unsuccessfully or delayed by the mail server. If the property is set to False, no delivery status information is sent back to the sender.

Note that delivery status notification is not available on all servers. It is also important to note that a message indicating that delivery was successful does not mean that the message was actually read by the recipient, only that it was delivered to their mailbox.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.Secure Property

Gets and sets a value which specifies if a secure connection is established.

[Visual Basic]

```
Public Property Secure As Boolean
```

[C#]

```
public bool Secure {get; set;}
```

Property Value

Returns **true** if a secure connection is established; otherwise returns **false**. The default value is **false**.

Remarks

The **Secure** property determines if a secure connection is established with the remote host. The default value for this property is **false**, which specifies that a standard connection to the server is used. To establish a secure connection, the application should set this property value to **true** prior to calling the **Connect** method. Once the connection has been established, the client may exchange data with the server as with standard connections.

It is strongly recommended that any application that sets this property **true** use error handling to trap an errors that may occur. If the control is unable to initialize the security libraries, or otherwise cannot create a secure session for the client, an exception may be generated when this property value is set.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.SecureCipher Property

Gets a value that specifies the encryption algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureCipher As SecureCipherAlgorithm
```

[C#]

```
public SmtpClient.SecureCipherAlgorithm SecureCipher {get;}
```

Property Value

A SecureCipherAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureCipher** property returns a value which identifies the algorithm used to encrypt the data stream. If a secure connection has not been established, this property will return a value of **cipherNone**.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.SecureHash Property

Gets a value that specifies the message digest algorithm used for a secure connection.

[Visual Basic]
```
Public ReadOnly Property SecureHash As SecureHashAlgorithm
```

[C#]
```
public SmtpClient.SecureHashAlgorithm SecureHash {get;}
```

Property Value

A SecureHashAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureHash** property returns a value which identifies the message digest (hash) algorithm that was selected when a secure connection was established. If a secure connection has not been established, this property will return a value of **hashNone**.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.SecureKeyExchange Property

Gets a value that specifies the key exchange algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureKeyExchange As SecureKeyAlgorithm
```

[C#]

```
public SmtpClient.SecureKeyAlgorithm SecureKeyExchange {get;}
```

Property Value

A SecureKeyAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureKeyExchange** property returns a value which identifies the key exchange algorithm that was selected when a secure connection was established. If a secure connection has not been established, this property will return a value of **keyExchangeNone**.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.SecureProtocol Property

Gets and sets a value which specifies the protocol used for a secure connection.

[Visual Basic]
```
Public Property SecureProtocol As SecurityProtocols
```

[C#]
```
public SmtpClient.SecurityProtocols SecureProtocol {get; set;}
```

Property Value

A SecurityProtocols enumeration value which identifies the protocol to be used when establishing a secure connection.

Remarks

The **SecureProtocol** property can be used to specify the security protocol to be used when establishing a secure connection with a server or accepting a secure connection from a client. By default, the class will attempt to use either SSL v3 or TLS v1 to establish the connection, with the appropriate protocol automatically selected based on the capabilities of the remote host. It is recommended that you only change this property value if you fully understand the implications of doing so. Assigning a value to this property will override the default protocol and force the class to attempt to use only the protocol specified.

Multiple security protocols may be specified by combining them using a bitwise **or** operator. After a connection has been established, this property will identify the protocol that was selected. Attempting to set this property after a connection has been established will result in an exception being thrown. This property should only be set after setting the **Secure** property to **true** and before calling the **Accept** or **Connect** methods.

In some cases, a server may only accept a secure connection if the TLS v1 protocol is specified. If the security protocol is not compatible with the server, then the connection will fail with an error indicating that the control is unable to establish a security context for the session. In this case, try assigning the property to **protocolTLS1** and attempt the connection again.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.Sender Property

Gets and sets the sender email address.

[Visual Basic]
```
Public Property Sender As String
```

[C#]
```
public string Sender {get; set;}
```

Property Value

A string which specifies the sender email address.

Remarks

The **Sender** property is used to specify the default address for the user who is sending the email message. This property should specify a valid email address in the standard Internet format. Typically this is the same address as specified in the From header field for the message, but it is not required that they be the same value.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.Status Property

Gets a value which specifies the current status of the client.

[Visual Basic]

```
Public ReadOnly Property Status As SmtpStatus
```

[C#]

```
public SmtpClient.SmtpStatus Status {get;}
```

Property Value

A SmtpStatus enumeration value which specifies the current client status.

Remarks

The **Status** property returns the current status of the client. This property can be used to check on blocking connections to determine if the client is interacting with the remote host before taking some action.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.ThreadModel Property

Gets and sets a value which specifies the threading model for the class instance.

[Visual Basic]
```
Public Property ThreadModel As ThreadingModel
```

[C#]
```
public SmtpClient.ThreadingModel ThreadModel {get; set;}
```

Property Value

Returns one or more ThreadingModel enumeration value which specifies the threading model for the client. The default value for this property is **modelSingleThread**.

Remarks

The **ThreadModel** property specifies the threading model that is used by the class instance when a connection is established. The default value for this property is **modelSingleThread**, which specifies that only the thread that established the connection should be permitted to invoke methods. It is important to note that this threading model does not limit the application to a single thread of execution. When a session is established using the **Connect** method, that session is attached to the thread that created it. From that point on, until the session is terminated, only the owner may invoke methods in that instance of the class. The ownership of the class instance may be transferred from one thread to another using the **AttachThread** method.

Setting this property to **modelFreeThread** disables certain internal safety checks that are performed by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads. The application must ensure that no two threads will attempt to invoke a blocking method at the same time. In other words, if one thread invokes a method, the application must ensure that another thread will not attempt to invoke any other method at the same time using the same instance of the class.

Changing the value of this property will not affect an active client session. The threading model must be specified prior to invoking the **Connect** method.

See Also

SmtpClient Class | SocketTools Namespace | AttachThread Method | ThreadingModel Enumeration | ThreadModel Attribute

SmtpClient.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]
```
Public Property ThrowError As Boolean
```

[C#]
```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.Timeout Property

Gets and sets a value which specifies a timeout period in seconds.

[Visual Basic]

```
Public Property Timeout As Integer
```

[C#]

```
public int Timeout {get; set;}
```

Property Value

An integer value which specifies a timeout period in seconds.

Remarks

Setting the **Timeout** property specifies the number of seconds until a blocking operation fails and returns an error.

The timeout period is only used when the client is in blocking mode. Although this property can be changed when the client is in non-blocking mode, the value will be ignored until the client is returned to blocking mode.

For most applications it is recommended the timeout period be set between 10 and 20 seconds.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.TimeZone Property

Gets and sets the current timezone offset in seconds.

[Visual Basic]

```
Public Property TimeZone As Integer
```

[C#]

```
public int TimeZone {get; set;}
```

Property Value

An integer value which specifies the current timezone offset in seconds.

Remarks

The **TimeZone** property returns the current offset from UTC in seconds. Setting the property changes the current timezone offset to the specified value. The value of this property is initially determined by the date and time settings on the local system.

This property value is used in conjunction with the **Localize** property to control how date and time localization is handled.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.Trace Property

Gets and sets a value which indicates if network function logging is enabled.

[Visual Basic]

```
Public Property Trace As Boolean
```

[C#]

```
public bool Trace {get; set;}
```

Property Value

Returns **true** if network function tracing is enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Trace** property is used to enable (or disable) the tracing of network function calls. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the SocketTools networking classes and is enabled or disabled for an entire process. This means that once trace logging is enabled for a given component, all of the function calls made by the process using any of the SocketTools classes will be logged. For example, if you have an application using both the File Transfer Protocol and Post Office Protocol classes, and you set the **Trace** property to **true**, function calls made by both classes will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all classes used by the process.

If trace logging is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the logfile is fairly large. Since logfiles can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

When redistributing your application, make sure that you include the **SocketTools10.TraceLog.dll** module with your installation. If this library is not present, then no trace output will be generated and the value of the **Trace** property will be ignored. Only those function calls made by the SocketTools networking classes will be logged. Calls made directly to the Windows Sockets API, or calls made by other classes, will not be logged.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.TraceFile Property

Gets and sets a value which specifies the name of the logfile.

[Visual Basic]

```
Public Property TraceFile As String
```

[C#]

```
public string TraceFile {get; set;}
```

Property Value

A string which specifies the name of the file.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named **SocketTools.log** is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFile** property should be set to the same value for each instance. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
 MyApp INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0 MyApp WRN:
connect(46, 192.0.0.1:1234, 16) returned -1 [10035] MyApp ERR: accept(46,
NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the control is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Note that if the specified file cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.TraceFlags Property

Gets and sets a value which specifies the client function tracing flags.

[Visual Basic]

```
Public Property TraceFlags As TraceOptions
```

[C#]

```
public SmtpClient.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration which specifies the amount of detail written to the trace logfile.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Because network function tracing is enabled per-process, the trace flags are shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFlags** property should be set to the same value for each instance. Changing the trace flags for any one instance of the class will affect the logging performed for all SocketTools classes used by the application.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being written and the error **errorOperationWouldBlock** occurs, a warning is generated because the application simply needs to attempt to write the data at a later time.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.UserName Property

Gets and sets the username used to authenticate the client session.

[Visual Basic]

```
Public Property UserName As String
```

[C#]

```
public string UserName {get; set;}
```

Property Value

A string which specifies the username.

Remarks

If a username is not specified when the **Connect** method is called, the value of this property will be used as the default username when establishing a connection with the server.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.Version Property

Gets a value which returns the current version of the SmtpClient class library.

[Visual Basic]

```
Public ReadOnly Property Version As String
```

[C#]

```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the SmtpClient class library. This value can be used by an application for validation and debugging purposes.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient Methods

The methods of the **SmtpClient** class are listed below. For a complete list of **SmtpClient** class members, see the SmtpClient Members topic.

Public Instance Methods

AddRecipient	Add an address to the recipient list for the current message.
AppendMessage	Overloaded. Append text to the current message being composed.
AttachThread	Attach an instance of the class to the current thread
Authenticate	Overloaded. Authenticate the client session with a username and password.
Cancel	Cancel the current blocking client operation.
CloseMessage	Closes the current message.
Command	Overloaded. Send a custom command to the server.
Connect	Overloaded. Establish a connection with a remote host.
CreateMessage	Overloaded. Begin the composition of a new message to be delivered.
Disconnect	Terminate the connection with a remote host.
Dispose	Overloaded. Releases all resources used by SmtpClient.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
ExpandAddress	Expand the specified email address.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the SmtpClient class.
Reset	Reset the internal state of the object, resetting all properties to their default values.
SendMessage	Overloaded. Submit the specified message to the mail server for delivery.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.

VerifyAddress	Verify the specified email address.
Write	Overloaded. Write one or more bytes of data to the server.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the SmtpClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.Authenticate Method

Authenticate the client session.

Overload List

Authenticate the client session.

public bool Authenticate();

Authenticate the client session with a username and password.

public bool Authenticate(string,string);

See Also

SmtpClient Class | SocketTools Namespace | Authentication Property | BearerToken Property | Extended Property | Password Property | UserName Property

SmtpClient.Authenticate Method ()

Authenticate the client session.

[Visual Basic]
```
Overloads Public Function Authenticate() As Boolean
```

[C#]
```
public bool Authenticate();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Authenticate** method is used to authenticate the current client session to the mail server, ensuring that only valid users may deliver messages through the server. This method uses the LOGIN authentication mechanism by default and you can specify an alternate authentication method by setting the the **Authentication** property.

Authentication requires the server to support the AUTH extended SMTP command and the **Extended** property must be set to True prior to establishing the connection. If the server does not support the specified type of authentication, an error will be returned.

The value of the **UserName** property is used to specify the username and the value of the **Password** property is used to specify the password. If OAuth 2.0 is being used for authentication, the value of the **BearerToken** property will be provided to the server.

If you with to use OAuth 2.0 for authentication, set the **Authentication** property to the desired authentication type and the **BearerToken** property to the value of the bearer token prior to calling this method. The connection must be secure, and the server must advertise its support for OAuth 2.0 or the authentication attempt will fail. This method will not attempt to automatically refresh an expired token.

If you provide a user name and password to the **Connect** method, or you set the **UserName** property and either the **Password** or **BearerToken** property prior to calling the **Connect** method, authentication will be automatically attempted at the time the connection is made. This method is only required if you do not provde user credentials when the connection is established and wish to authenticate the client session at a later time.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.Authenticate Overload List | Authentication Property | BearerToken Property | Extended Property | Password Property | UserName Property

SmtpClient.Authenticate Method (String, String)

Authenticate the client session with a username and password.

[Visual Basic]

```
Overloads Public Function Authenticate( _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]

```
public bool Authenticate(
   string userName,
   string userPassword
);
```

Parameters

userName

A string which specifies the username used to authenticate the client session.

userPassword

A string which specifies the password which will be used to authenticate the client session with the remote host. Not all mail servers require the client to authenticate the session. If you are using OAuth 2.0 authentication, this parameter specifies the bearer token provided by the mail service.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Authenticate** method is used to authenticate the current client session to the mail server, ensuring that only valid users may deliver messages through the server. This method uses the LOGIN authentication mechanism by default and you can specify an alternate authentication method by setting the the **Authentication** property.

Authentication requires the server to support the AUTH extended SMTP command and the **Extended** property must be set to True prior to establishing the connection. If the server does not support the specified type of authentication, an error will be returned.

If you with to use OAuth 2.0 for authentication, set the **Authentication** property to the desired authentication type prior to calling this method. The connection must be secure, and the server must advertise its support for OAuth 2.0 or the authentication attempt will fail. This method will not attempt to automatically refresh an expired token.

If you provide a user name and password to the **Connect** method, or you set the **UserName** property and either the **Password** or **BearerToken** property prior to calling the **Connect** method, authentication will be automatically attempted at the time the connection is made. This method is only required if you do not provde user credentials when the connection is established and wish to authenticate the client session at a later time.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.Authenticate Overload List | Authentication Property | BearerToken Property | Extended Property | Password Property | UserName Property

SmtpClient.AddRecipient Method

Add an address to the recipient list for the current message.

[Visual Basic]

```
Public Function AddRecipient( _
   ByVal address As String _
) As Boolean
```

[C#]

```
public bool AddRecipient(
   string address
);
```

Parameters

address

A string which specifies the recipient address.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AddRecipient** method adds the specified address to the recipient list for the current message. This method should be called after the message transaction has begun with a call to the **CreateMessage** method. Most servers impose a limit of approximately 100 recipient addresses that will be accepted for a single message.

Note that this method does not update the **Recipient** property, which maintains an internal list of recipient addresses used by the **SendMessage** method. This method should only be used in conjunction with the **CreateMessage** method.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.AppendMessage Method

Append text to the current message being composed.

Overload List

Append text to the current message being composed.

public bool AppendMessage(byte[],int);

Append text to the current message being composed.

public bool AppendMessage(string);

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.AppendMessage Method (Byte[], Int32)

Append text to the current message being composed.

[Visual Basic]
```
Overloads Public Function AppendMessage( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Boolean
```

[C#]
```
public bool AppendMessage(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array which specifies the data to be appended to the message.

length

An integer value which specifies the maximum number of bytes of data to append to the message. This value cannot be larger than the size of the buffer specified by the caller

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AppendMessage** method appends the specified text to the current message. This method will cause the current thread to block until the article transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

This method is useful for composing a message where the application needs to dynamically create the header, followed by a large amount of text. The message contents should be text, with each line terminated with a carriage return and linefeed character. Not all mail servers support sending 8-bit characters, so the message contents may need to be encoded if it uses anything other than standard US ASCII. To append binary data, it should be encoded using either the uucode or base64 (MIME) algorithms. It is recommended that you use the **SocketTools.MailMessage** class to manage file attachments and other complex message types.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.AppendMessage Overload List

SmtpClient.AppendMessage Method (String)

Append text to the current message being composed.

[Visual Basic]
```
Overloads Public Function AppendMessage( _
   ByVal buffer As String _
) As Boolean
```

[C#]
```
public bool AppendMessage(
   string buffer
);
```

Parameters

buffer
A byte array which specifies the data to be appended to the message.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **AppendMessage** method appends the specified text to the current message. This method will cause the current thread to block until the article transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

This method is useful for composing a message where the application needs to dynamically create the header, followed by a large amount of text. The message contents should be text, with each line terminated with a carriage return and linefeed character. Not all mail servers support sending 8-bit characters, so the message contents may need to be encoded if it uses anything other than standard US ASCII. To append binary data, it should be encoded using either the uucode or base64 (MIME) algorithms. It is recommended that you use the **SocketTools.MailMessage** class to manage file attachments and other complex message types.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.AppendMessage Overload List

SmtpClient.AttachThread Method

Attach an instance of the class to the current thread

[Visual Basic]
```
Public Function AttachThread() As Boolean
```

[C#]
```
public bool AttachThread();
```

Return Value

A boolean value which specifies if the client could be attached to the current thread. If this method returns **false**, the client could not be attached to the thread and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

When an instance of the class is created it is associated with the current thread that created it. Normally, if another thread attempts to perform an operation using that instance, an error is returned. This is used to ensure that other threads cannot interfere with an operation being performed by the owner thread. In some cases, it may be desirable for one thread in an application to create an instance of the class, establish a connection and then pass that instance to another worker thread. The **AttachThread** method can be used to change the ownership of the class instance to the new worker thread.

This method should be called by the new thread immediately after it has been created, and if the new thread does not release the handle itself, the ownership of the handle should be restored by the original thread. Under no circumstances should **AttachThread** be used to forcibly destroy an instance of a class allocated by another thread while a blocking operation is in progress. To cancel a blocking operation, use the **Cancel** method and then delete the class instance after the blocking function exits and control is returned to the current thread.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.Cancel Method

Cancel the current blocking client operation.

[Visual Basic]
```
Public Sub Cancel()
```

[C#]
```
public void Cancel();
```

Remarks

When the **Cancel** method is called, the blocking client operation will not immediately fail. An internal flag is set which causes the blocking operation to exit with an error. This means that the application cannot cancel an operation and immediately perform some other blocking function. Instead it must allow the calling stack to unwind, returning back to the blocking operation before making any further function calls.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.CloseMessage Method

Closes the current message.

[Visual Basic]
```
Public Function CloseMessage() As Boolean
```

[C#]
```
public bool CloseMessage();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CloseArticle** method closes the current message that has been created and submits it to the mail server for delivery.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.Command Method

Send a custom command to the server.

Overload List

Send a custom command to the server.

public bool Command(string);

Send a custom command to the server.

public bool Command(string,string);

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.Command Method (String)

Send a custom command to the server.

[Visual Basic]

```
Overloads Public Function Command( _
   ByVal command As String _
) As Boolean
```

[C#]

```
public bool Command(
   string command
);
```

Parameters

command

A string which specifies the command to send. Valid commands vary based on the Internet protocol and the type of server that the client is connected to. Consult the protocol standard and/or the technical reference documentation for the server to determine what commands may be issued by a client application.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Command** method sends a command to the remote host and processes the result code sent back in response to that command. This method can be used to send custom commands to a server to take advantage of features or capabilities that may not be supported internally by the class library.

To determine the specific status code returned by the server, check the value of the **ResultCode** property after the method returns.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.Command Overload List

SmtpClient.Command Method (String, String)

Send a custom command to the server.

[Visual Basic]
```
Overloads Public Function Command( _
   ByVal command As String, _
   ByVal parameters As String _
) As Boolean
```

[C#]
```
public bool Command(
   string command,
   string parameters
);
```

Parameters

command

A string which specifies the command to send. Valid commands vary based on the Internet protocol and the type of server that the client is connected to. Consult the protocol standard and/or the technical reference documentation for the server to determine what commands may be issued by a client application.

parameters

An string which specifies one or more parameters to be sent along with the command. If more than one parameter is required, they must be separated by a single space character. Consult the protocol standard and/or technical reference documentation for the server to determine what parameters should be provided when issuing a specific command.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Command** method sends a command to the remote host and processes the result code sent back in response to that command. This method can be used to send custom commands to a server to take advantage of features or capabilities that may not be supported internally by the class library.

To determine the specific status code returned by the server, check the value of the **ResultCode** property after the method returns.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.Command Overload List

SmtpClient.Connect Method

Establish a connection with a remote host.

Overload List

Establish a connection with a remote host.

public bool Connect();

Establish a connection with a remote host.

public bool Connect(string);

Establish a connection with a remote host.

public bool Connect(string,int);

Establish a connection with a remote host.

public bool Connect(string,int,int);

Establish a connection with a remote host.

public bool Connect(string,int,int,SmtpOptions);

Establish a connection with a remote host.

public bool Connect(string,int,string,string);

Establish a connection with a remote host.

public bool Connect(string,int,string,string,int);

Establish a connection with a remote host.

public bool Connect(string,int,string,string,int,SmtpOptions);

Establish a connection with a remote host.

public bool Connect(string,string,string);

See Also

SmtpClient Class | SocketTools Namespace | Authentication Property | BearerToken Property | HostName Property | Options Property | Password Property | UserName Property

SmtpClient.Connect Method ()

Establish a connection with a remote host.

[Visual Basic]
```
Overloads Public Function Connect() As Boolean
```

[C#]
```
public bool Connect();
```

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection using assigned property values.

The value of the **HostName** or **HostAddress** property will be used to determine the host name or address to connect to.

The value of the **RemotePort** property will be used to determine the port number to connect to.

The value of the **UserName** property specifies the username that will be used to authenticate the session.

The value of the **Password** property specifies the password that will be used to authenticate the session.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.Connect Overload List | Authentication Property | BearerToken Property | HostName Property | Options Property | Password Property | UserName Property

SmtpClient.Connect Method (String)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection using assigned property values.

The value of the **RemotePort** property will be used to determine the port number to connect to.

The value of the **UserName** property specifies the username that will be used to authenticate the session.

The value of the **Password** property specifies the password that will be used to authenticate the session.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.Connect Overload List | Authentication Property | BearerToken Property | HostName Property | Options Property | Password Property | UserName Property

SmtpClient.Connect Method (String, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection using assigned property values.

The value of the **UserName** property specifies the username that will be used to authenticate the session.

The value of the **Password** property specifies the password that will be used to authenticate the session.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.Connect Overload List | Authentication Property | BearerToken Property | HostName Property | Options Property | Password Property | UserName Property

SmtpClient.Connect Method (String, Int32, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   int timeout
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection using assigned property values.

The value of the **UserName** property specifies the username that will be used to authenticate the session.

The value of the **Password** property specifies the password that will be used to authenticate the session.

The value of the **Options** property will be used to specify the default options for the connection.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.Connect Overload List | Authentication Property | BearerToken Property | HostName Property | Options Property | Password Property | UserName Property

SmtpClient.Connect Method (String, Int32, Int32, SmtpOptions)

Establish a connection with a remote host.

[Visual Basic]
```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer, _
   ByVal options As SmtpOptions _
) As Boolean
```

[C#]
```
public bool Connect(
   string hostName,
   int hostPort,
   int timeout,
   SmtpOptions options
);
```

Parameters

hostName
A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort
An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

timeout
An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

options
One or more of the SmtpOptions enumeration flags.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.Connect Overload List | Authentication Property | BearerToken Property | HostName Property | Options Property | Password Property | UserName Property

SmtpClient.Connect Method (String, Int32, String, String)

Establish a connection with a remote host.

[Visual Basic]
```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]
```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName

A string which specifies the username which will be used to authenticate the client session with the remote host. Not all mail servers require the client to authenticate the session.

userPassword

A string which specifies the password which will be used to authenticate the client session with the remote host. Not all mail servers require the client to authenticate the session. If you are using OAuth 2.0 authentication, this parameter specifies the bearer token provided by the mail service.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection using assigned property values.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.Connect Overload List | Authentication Property | BearerToken Property | HostName Property | Options Property | Password Property | UserName Property

SmtpClient.Connect Method (String, Int32, String, String, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   int timeout
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName

A string which specifies the username which will be used to authenticate the client session with the remote host. Not all mail servers require the client to authenticate the session.

userPassword

A string which specifies the password which will be used to authenticate the client session with the remote host. Not all mail servers require the client to authenticate the session. If you are using OAuth 2.0 authentication, this parameter specifies the bearer token provided by the mail service.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.Connect Overload List | Authentication Property | BearerToken Property | HostName Property | Options Property | Password Property | UserName Property

SmtpClient.Connect Method (String, Int32, String, String, Int32, SmtpOptions)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal userPassword As String, _
   ByVal timeout As Integer, _
   ByVal options As SmtpOptions _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string userPassword,
   int timeout,
   SmtpOptions options
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName

A string which specifies the username which will be used to authenticate the client session with the remote host. Not all mail servers require the client to authenticate the session.

userPassword

A string which specifies the password which will be used to authenticate the client session with the remote host. Not all mail servers require the client to authenticate the session. If you are using OAuth 2.0 authentication, this parameter specifies the bearer token provided by the mail service.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

options

One or more of the SmtpOptions enumeration flags.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.Connect Overload List | Authentication Property | BearerToken Property | HostName Property | Options Property | Password Property | UserName Property

SmtpClient.Connect Method (String, String, String)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   string userName,
   string userPassword
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

userName

A string which specifies the username which will be used to authenticate the client session with the remote host. Not all mail servers require the client to authenticate the session.

userPassword

A string which specifies the password which will be used to authenticate the client session with the remote host. Not all mail servers require the client to authenticate the session. If you are using OAuth 2.0 authentication, this parameter specifies the bearer token provided by the mail service.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection using assigned property values.

The value of the **RemotePort** property will be used to determine the port number to connect to.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.Connect Overload List | Authentication Property | BearerToken Property | HostName Property | Options Property | Password Property | UserName Property

SmtpClient.CreateMessage Method

Begin the composition of a new message to be delivered.

Overload List

Begin the composition of a new message to be delivered.

public bool CreateMessage(string);

Begin the composition of a new message to be delivered.

public bool CreateMessage(string,int);

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.CreateMessage Method (String)

Begin the composition of a new message to be delivered.

```
[Visual Basic]
Overloads Public Function CreateMessage( _
   ByVal messageSender As String _
) As Boolean
```

```
[C#]
public bool CreateMessage(
   string messageSender
);
```

Parameters

messageSender

A string which specifies the email address of the user sending the message. This typically corresponds to the address in the From header of the message, but it is not required that they be the same.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateMessage** method begins the composition of a new message to be submitted to the mail server for delivery. There are several steps that must be followed when dynamically composing a message using the **CreateMessage** method:

1. Call the **CreateMessage** method to begin the message composition. The sender email address should generally be the same address as the one used in the "From" header field in the message.
2. Call the **AddRecipient** method for each recipient of the message. These addresses are typically specified in the "To" and "Cc" header fields in the message. Additional addresses may also be be provided which are not specified in the email message itself. This is how one or more blind carbon copies of a message is delivered. Most servers have a limit on the total number of recipients that may be specified for a single message. This limit is usually around 100 addresses.
3. Call the **Write** method to write the contents of the message to the data stream. The application may also choose to use the **AppendMessage** method to write out a large amount of message data.
4. Call the **CloseMessage** method to close the message and submit it to the mail server for delivery.

For applications that do not need to dynamically compose the message and already have the message contents stored in a file or memory buffer, the **SendMessage** method is the preferred method of submitting a message for delivery

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.CreateMessage Overload List

SmtpClient.CreateMessage Method (String, Int32)

Begin the composition of a new message to be delivered.

[Visual Basic]

```
Overloads Public Function CreateMessage( _
   ByVal messageSender As String, _
   ByVal messageSize As Integer _
) As Boolean
```

[C#]

```
public bool CreateMessage(
   string messageSender,
   int messageSize
);
```

Parameters

messageSender

A string which specifies the email address of the user sending the message. This typically corresponds to the address in the From header of the message, but it is not required that they be the same.

messageSize

An integer which specifies the size of the message in bytes. If the size of the message is unknown, this argument should be omitted or passed as value of zero. This argument is ignored if the server does not support extended features. If the message size is larger than what the server will accept, this method will fail. Most Internet Service Providers impose a limit on the size of an email message, typically between 5 and 10 megabytes.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **CreateMessage** method begins the composition of a new message to be submitted to the mail server for delivery. There are several steps that must be followed when dynamically composing a message using the **CreateMessage** method:

1. Call the **CreateMessage** method to begin the message composition. The sender email address should generally be the same address as the one used in the "From" header field in the message.
2. Call the **AddRecipient** method for each recipient of the message. These addresses are typically specified in the "To" and "Cc" header fields in the message. Additional addresses may also be be provided which are not specified in the email message itself. This is how one or more blind carbon copies of a message is delivered. Most servers have a limit on the total number of recipients that may be specified for a single message. This limit is usually around 100 addresses.
3. Call the **Write** method to write the contents of the message to the data stream. The application may also choose to use the **AppendMessage** method to write out a large amount of message data.
4. Call the **CloseMessage** method to close the message and submit it to the

mail server for delivery.

For applications that do not need to dynamically compose the message and already have the message contents stored in a file or memory buffer, the **SendMessage** method is the preferred method of submitting a message for delivery

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.CreateMessage Overload List

SmtpClient.Disconnect Method

Terminate the connection with a remote host.

[Visual Basic]

```
Public Sub Disconnect()
```

[C#]

```
public void Disconnect();
```

Remarks

The **Disconnect** method terminates the connection with the remote host and releases the client handle allocated by the class. Note that the socket is not immediately released when the connection is terminated and will enter a wait state for two minutes. After the time wait period has elapsed, the client will be released by the operating system. This is a normal safety mechanism to handle any packets that may arrive after the connection has been closed.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.Dispose Method

Releases all resources used by SmtpClient.

Overload List

Releases all resources used by SmtpClient.

public void Dispose();

Releases the unmanaged resources allocated by the SmtpClient class and optionally releases the managed resources.

protected virtual void Dispose(bool);

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.Dispose Method ()

Releases all resources used by SmtpClient.

[Visual Basic]

```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]

```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.Dispose Overload List

SmtpClient.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the SmtpClient class and optionally releases the managed resources.

[Visual Basic]

```
Overridable Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]

```
protected virtual void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **SmtpClient** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.Dispose Overload List

SmtpClient.ExpandAddress Method

Expand the specified email address.

[Visual Basic]

```
Public Function ExpandAddress( _
   ByVal address As String, _
   ByRef expandedAddress As String _
) As Boolean
```

[C#]

```
public bool ExpandAddress(
   string address,
   ref string expandedAddress
);
```

Parameters

address

A string which specifies the address to expand.

expandedAddress

A string passed by reference which will contain the list of expanded addresses when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **ExpandAddress** method requests that the server expand the specified email address. Typically this is used to expand aliases which refer to a mailing list, returning all of the members of that list. A server may not support this command, or may restrict its usage. An application should not depend on the ability to expand addresses.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.Finalize Method

Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.

[Visual Basic]
```
Overrides Protected Sub Finalize()
```

[C#]
```
protected override void Finalize();
```

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.Initialize Method

Initialize an instance of the SmtpClient class.

Overload List

Initialize an instance of the SmtpClient class.

public bool Initialize();

Initialize an instance of the SmtpClient class.

public bool Initialize(string);

See Also

SmtpClient Class | SocketTools Namespace | Uninitialize Method

SmtpClient.Initialize Method ()

Initialize an instance of the SmtpClient class.

[Visual Basic]
```
Overloads Public Function Initialize() As Boolean
```

[C#]
```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the SmtpClient class, loading the networking library and allocating resources for the current thread. Typically it is not necessary to explicitly call this method because the instance of the class is initialized by the class constructor. However, if the **Uninitialize** method is called, the class must be re-initialized before any other methods are called.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.Initialize Overload List | Uninitialize Method

SmtpClient.Initialize Method (String)

Initialize an instance of the SmtpClient class.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the SmtpClient class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the SmtpClient class.

Example

The following example shows how to use the Initialize method to initialize an instance of the class. This example assumes that the license key string has been defined in code.

```
SocketTools.SmtpClient smtpClient = new SocketTools.SmtpClient();

if (smtpClient.Initialize(strLicenseKey) == false)
{
    MessageBox.Show(smtpClient.LastErrorString, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    return;
}
```

```
Dim smtpClient As New SocketTools.SmtpClient

If smtpClient.Initialize(strLicenseKey) = False Then
    MsgBox(smtpClient.LastErrorString, vbIconExclamation)
    Exit Sub
End If
```

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

SmtpClient.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. If a socket has been allocated, it will be released and any active connections will be terminated. All properties will be reset to their default values.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.SendMessage Method

Submit the specified message to the mail server for delivery.

Overload List

Submit the specified message to the mail server for delivery.

public bool SendMessage(byte[],int);

Submit the specified message to the mail server for delivery.

public bool SendMessage(string);

Submit the specified message to the mail server for delivery.

public bool SendMessage(string,string,byte[],int);

Submit the specified message to the mail server for delivery.

public bool SendMessage(string,string,string);

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.SendMessage Method (Byte[], Int32)

Submit the specified message to the mail server for delivery.

[Visual Basic]

```
Overloads Public Function SendMessage( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool SendMessage(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array that contains the message to be delivered to the specified recipients. The message must be text and conform to the basic structure defined in RFC 822. There must be one or more headers separated by a blank line, followed by the body of the message. Each line of text must be terminated by a carriage return and linefeed character sequence. Note that more complex multipart MIME messages may also be used, but it is recommended that you use the **SocketTools.MailMessage** class to compose them.

length

An integer value which specifies the number of bytes of data in the specified buffer. This value cannot be larger than the size of the buffer provided by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SendMessage** method enables an application to send a formatted email message using the current mail server. This provides a convenient one-step method of addressing and sending a message, and is designed to easily integrate with the **SocketTools.MailMessage** class.

This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The value of the **Sender** property will be used to specify the address of the user sending the message. The addresses assigned to the **Recipient** array will be used to specify the message recipients.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.SendMessage Overload List

SmtpClient.SendMessage Method (String)

Submit the specified message to the mail server for delivery.

[Visual Basic]
```
Overloads Public Function SendMessage( _
   ByVal buffer As String _
) As Boolean
```

[C#]
```
public bool SendMessage(
   string buffer
);
```

Parameters

buffer

A byte array that contains the message to be delivered to the specified recipients. The message must be text and conform to the basic structure defined in RFC 822. There must be one or more headers separated by a blank line, followed by the body of the message. Each line of text must be terminated by a carriage return and linefeed character sequence. Note that more complex multipart MIME messages may also be used, but it is recommended that you use the **SocketTools.MailMessage** class to compose them.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SendMessage** method enables an application to send a formatted email message using the current mail server. This provides a convenient one-step method of addressing and sending a message, and is designed to easily integrate with the **SocketTools.MailMessage** class.

This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

The value of the **Sender** property will be used to specify the address of the user sending the message. The addresses assigned to the **Recipient** array will be used to specify the message recipients.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.SendMessage Overload List

SmtpClient.SendMessage Method (String, String, Byte[], Int32)

Submit the specified message to the mail server for delivery.

[Visual Basic]

```
Overloads Public Function SendMessage( _
   ByVal senderAddress As String, _
   ByVal recipientAddress As String, _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool SendMessage(
   string senderAddress,
   string recipientAddress,
   byte[] buffer,
   int length
);
```

Parameters

senderAddress
A string argument which specifies the email address of the person sending the message. This typically corresponds to the address in the From header of the message, but it is not required that they be the same.

recipientAddress
A string argument which specifies the email address of the person or persons to receive the message. Multiple addresses may be specified by separating each address with a comma. It should be noted that this protocol is only concerned with the delivery of a message and not its contents. Header fields in the message are not parsed to automatically determine the recipients. This argument should be a concatenation of all recipients, including carbon copies and blind carbon copies, with each address separated with a comma.

buffer
A byte array that contains the message to be delivered to the specified recipients. The message must be text and conform to the basic structure defined in RFC 822. There must be one or more headers separated by a blank line, followed by the body of the message. Each line of text must be terminated by a carriage return and linefeed character sequence. Note that more complex multipart MIME messages may also be used, but it is recommended that you use the **SocketTools.MailMessage** class to compose them.

length
An integer value which specifies the number of bytes of data in the specified buffer. This value cannot be larger than the size of the buffer provided by the caller.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SendMessage** method enables an application to send a formatted email message using the current mail server. This provides a convenient one-step method of addressing and sending a message, and is designed to easily integrate with the **SocketTools.MailMessage** class.

This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.SendMessage Overload List

SmtpClient.SendMessage Method (String, String, String)

Submit the specified message to the mail server for delivery.

[Visual Basic]

```
Overloads Public Function SendMessage( _
   ByVal senderAddress As String, _
   ByVal recipientAddress As String, _
   ByVal buffer As String _
) As Boolean
```

[C#]

```
public bool SendMessage(
   string senderAddress,
   string recipientAddress,
   string buffer
);
```

Parameters

senderAddress

A string argument which specifies the email address of the person sending the message. This typically corresponds to the address in the From header of the message, but it is not required that they be the same.

recipientAddress

A string argument which specifies the email address of the person or persons to receive the message. Multiple addresses may be specified by separating each address with a comma. It should be noted that this protocol is only concerned with the delivery of a message and not its contents. Header fields in the message are not parsed to automatically determine the recipients. This argument should be a concatenation of all recipients, including carbon copies and blind carbon copies, with each address separated with a comma.

buffer

A string that contains the message to be delivered to the specified recipients. The message must be text and conform to the basic structure defined in RFC 822. There must be one or more headers separated by a blank line, followed by the body of the message. Each line of text must be terminated by a carriage return and linefeed character sequence. Note that more complex multipart MIME messages may also be used, but it is recommended that you use the **SocketTools.MailMessage** class to compose them.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SendMessage** method enables an application to send a formatted email message using the current mail server. This provides a convenient one-step method of addressing and sending a message, and is designed to easily integrate with the **SocketTools.MailMessage** class.

This method will cause the current thread to block until the message transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.SendMessage Overload List

SmtpClient.Uninitialize Method

Uninitialize the class library and release any resources allocated for the current thread.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method terminates any active connection, releases resources allocated for the current thread and unloads the networking library. After this method has been called, no further client operations may be performed until the class instance has been re-initialized.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the class is no longer needed.

See Also

SmtpClient Class | SocketTools Namespace | Initialize Method

SmtpClient.VerifyAddress Method

Verify the specified email address.

```
[Visual Basic]
Public Function VerifyAddress( _
   ByVal address As String, _
   ByRef verifiedAddress As String _
) As Boolean
```

```
[C#]
public bool VerifyAddress(
   string address,
   ref string verifiedAddress
);
```

Parameters

address
A string which specifies the address to verify.

verifiedAddress
A string passed by reference which will contain the verified address when the method returns.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **VerifyAddress** method requests that the server verify the specified email address. Typically this is used to verify that a recipient address is valid, and return a fully qualified email address for that recipient. A server may not support this command, or may restrict its usage. An application should not depend on the ability to verify addresses.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.Write Method

Write one or more bytes of data to the server.

Overload List

Write one or more bytes of data to the server.

public int Write(byte[]);

Write one or more bytes of data to the server.

public int Write(byte[],int);

Write a string of characters to the server.

public int Write(string);

Write a string of characters to the server.

public int Write(string,int);

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.Write Method (Byte[])

Write one or more bytes of data to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Write(
   byte[] buffer
);
```

Parameters

buffer
A byte array that contains the data to be written to the server.

Return Value

An integer value which specifies the number of bytes actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

If the **Write** method is used to send the message contents to the server, the application must first call the **CreateMessage** method to specify the sender and the length of the message, followed by one or more calls to the **AddRecipient** method to specify each recipient of the message. When all of the message text has been submitted to the server, the application must call the **CloseMessage** method.

The message text is filtered by the **Write** method, and it will automatically normalize end-of-line character sequences to ensure the message meets the protocol requirements. The message itself must be in a standard RFC 822 or multi-part MIME message format, or the server may reject the message. Binary data, such as file attachments, should always be encoded. The **SocketTools.MailMessage** class can be used to compose and export a message in the correct format, which can then be submitted to the server.

It is recommended that most applications use the **SendMessage** method, which submits the message in a single method call.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.Write Overload List

SmtpClient.Write Method (Byte[], Int32)

Write one or more bytes of data to the server.

```
[Visual Basic]
Overloads Public Function Write( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

```
[C#]
public int Write(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array that contains the data to be written to the server.

length

An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

If the **Write** method is used to send the message contents to the server, the application must first call the **CreateMessage** method to specify the sender and the length of the message, followed by one or more calls to the **AddRecipient** method to specify each recipient of the message. When all of the message text has been submitted to the server, the application must call the **CloseMessage** method.

The message text is filtered by the **Write** method, and it will automatically normalize end-of-line character sequences to ensure the message meets the protocol requirements. The message itself must be in a standard RFC 822 or multi-part MIME message format, or the server may reject the message. Binary data, such as file attachments, should always be encoded. The **SocketTools.MailMessage** class can be used to compose and export a message in the correct format, which can then be submitted to the server.

It is recommended that most applications use the **SendMessage** method, which submits the message in a single method call.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.Write Overload List

SmtpClient.Write Method (String)

Write a string of characters to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String _
) As Integer
```

[C#]

```
public int Write(
   string buffer
);
```

Parameters

buffer

A string which contains the data to be written to the server.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

If the **Write** method is used to send the message contents to the server, the application must first call the **CreateMessage** method to specify the sender and the length of the message, followed by one or more calls to the **AddRecipient** method to specify each recipient of the message. When all of the message text has been submitted to the server, the application must call the **CloseMessage** method.

The message text is filtered by the **Write** method, and it will automatically normalize end-of-line character sequences to ensure the message meets the protocol requirements. The message itself must be in a standard RFC 822 or multi-part MIME message format, or the server may reject the message. Binary data, such as file attachments, should always be encoded. The **SocketTools.MailMessage** class can be used to compose and export a message in the correct format, which can then be submitted to the server.

It is recommended that most applications use the **SendMessage** method, which submits the message in a single method call.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.Write Overload List

SmtpClient.Write Method (String, Int32)

Write a string of characters to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Write(
   string buffer,
   int length
);
```

Parameters

buffer
A string which contains the data to be written to the server.

length
An integer value which specifies the maximum number of characters to write. This value cannot be larger than the length of the string specified by the caller.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

If the **Write** method is used to send the message contents to the server, the application must first call the **CreateMessage** method to specify the sender and the length of the message, followed by one or more calls to the **AddRecipient** method to specify each recipient of the message. When all of the message text has been submitted to the server, the application must call the **CloseMessage** method.

The message text is filtered by the **Write** method, and it will automatically normalize end-of-line character sequences to ensure the message meets the protocol requirements. The message itself must be in a standard RFC 822 or multi-part MIME message format, or the server may reject the message. Binary data, such as file attachments, should always be encoded. The **SocketTools.MailMessage** class can be used to compose and export a message in the correct format, which can then be submitted to the server.

It is recommended that most applications use the **SendMessage** method, which submits the message in a single method call.

See Also

SmtpClient Class | SocketTools Namespace | SmtpClient.Write Overload List

SmtpClient Events

The events of the **SmtpClient** class are listed below. For a complete list of **SmtpClient** class members, see the SmtpClient Members topic.

Public Instance Events

⚡ OnCancel	Occurs when a blocking client operation is canceled.
⚡ OnCommand	Occurs when the client sends a command to the remote host and receives a reply indicating the result of that command.
⚡ OnConnect	Occurs when a connection is established with the remote host.
⚡ OnDisconnect	Occurs when the remote host disconnects from the local system.
⚡ OnError	Occurs when an client operation fails.
⚡ OnProgress	Occurs as a data stream is being read or written to the client.
⚡ OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.OnCancel Event

Occurs when a blocking client operation is canceled.

[Visual Basic]

```
Public Event OnCancel As EventHandler
```

[C#]

```
public event EventHandler OnCancel;
```

Remarks

The **OnCancel** event is generated when a blocking client operation, such as sending or receiving data, is canceled with the **Cancel** method. To assist in determining which operation was canceled, check the value of the **Status** property.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.OnCommand Event

Occurs when the client sends a command to the remote host and receives a reply indicating the result of that command.

[Visual Basic]
```
Public Event OnCommand As OnCommandEventHandler
```

[C#]
```
public event OnCommandEventHandler OnCommand;
```

Event Data

The event handler receives an argument of type SmtpClient.CommandEventArgs containing data related to this event. The following **SmtpClient.CommandEventArgs** properties provide information specific to this event.

Property	Description
ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.

Remarks

The **OnCommand** event is generated when the client receives a reply from the server after some action has been taken.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.CommandEventArgs Class

Provides data for the OnCommand event.

For a list of all members of this type, see SmtpClient.CommandEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.SmtpClient.CommandEventArgs

[Visual Basic]
```
Public Class SmtpClient.CommandEventArgs
    Inherits EventArgs
```

[C#]
```
public class SmtpClient.CommandEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

CommandEventArgs specifies the result code and result string for the last command executed by the server.

The OnCommand event occurs whenever a command is executed on the server.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SmtpClient (in SocketTools.SmtpClient.dll)

See Also

SmtpClient.CommandEventArgs Members | SocketTools Namespace

SmtpClient.CommandEventArgs Members

SmtpClient.CommandEventArgs overview

Public Instance Constructors

SmtpClient.CommandEventArgs Constructor	Initializes a new instance of the SmtpClient.CommandEventArgs class.

Public Instance Properties

ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

SmtpClient.CommandEventArgs Class | SocketTools Namespace

SmtpClient.CommandEventArgs Constructor

Initializes a new instance of the SmtpClient.CommandEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public SmtpClient.CommandEventArgs();
```

See Also

SmtpClient.CommandEventArgs Class | SocketTools Namespace

SmtpClient.CommandEventArgs Properties

The properties of the **SmtpClient.CommandEventArgs** class are listed below. For a complete list of **SmtpClient.CommandEventArgs** class members, see the SmtpClient.CommandEventArgs Members topic.

Public Instance Properties

ResultCode	Gets a value which specifies the last result code returned by the server.
ResultString	Gets a string value which describes the result of the previous command.

See Also

SmtpClient.CommandEventArgs Class | SocketTools Namespace

SmtpClient.CommandEventArgs.ResultCode Property

Gets a value which specifies the last result code returned by the server.

[Visual Basic]

```
Public ReadOnly Property ResultCode As Integer
```

[C#]

```
public int ResultCode {get;}
```

Property Value

An integer value which specifies the last result code returned by the server.

Remarks

This property should be checked after the **Command** method is used to execute a command on the server to determine if the operation was successful. Result codes are three-digit numeric values returned by the remote server and may be broken down into the following ranges:

ResultCode	Description
100-199	Positive preliminary result. This indicates that the requested action is being initiated, and the client should expect another reply from the server before proceeding.
200-299	Positive completion result. This indicates that the server has successfully completed the requested action.
300-399	Positive intermediate result. This indicates that the requested action cannot complete until additional information is provided to the server.
400-499	Transient negative completion result. This indicates that the requested action did not take place, but the error condition is temporary and may be attempted again.
500-599	Permanent negative completion result. This indicates that the requested action did not take place.

It is important to note that while some result codes have become standardized, not all servers respond to commands using the same result codes. For example, one server may respond with a result code of 221 to indicate success, while another may respond with a value of 235. It is recommended that applications check for ranges of values to determine if a command was successful, not a specific value.

See Also

SmtpClient.CommandEventArgs Class | SocketTools Namespace

SmtpClient.CommandEventArgs.ResultString Property

Gets a string value which describes the result of the previous command.

[Visual Basic]

```
Public ReadOnly Property ResultString As String
```

[C#]

```
public string ResultString {get;}
```

Property Value

A string which describes the result of the previous command executed on the server.

Remarks

This string is generated by the remote server, and typically is used to describe the result code. For example, if an error is indicated by the result code, the result string may describe the condition that caused the error.

See Also

SmtpClient.CommandEventArgs Class | SocketTools Namespace

SmtpClient.OnConnect Event

Occurs when a connection is established with the remote host.

[Visual Basic]

```
Public Event OnConnect As EventHandler
```

[C#]

```
public event EventHandler OnConnect;
```

Remarks

The **OnConnect** event occurs when a connection is made with a remote host as a result of a **Connect** method call. When the **Connect** method is called and the **Blocking** property is set to **false**, a socket is created but the connection is not actually established until after this event occurs. Between the time connection process is started and this event fires, no operation may be performed on the client other than calling the **Disconnect** method.

This event is only generated if the client is in non-blocking mode.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.OnDisconnect Event

Occurs when the remote host disconnects from the local system.

[Visual Basic]

```
Public Event OnDisconnect As EventHandler
```

[C#]

```
public event EventHandler OnDisconnect;
```

Remarks

The **OnDisconnect** event occurs when the remote host closes its connection, terminating the client session with the application. Because there may still be data in the client receive buffers, you should continue to read data from the client until the **Read** method returns a value of 0. Once all of the data has been read, you should call the **Disconnect** method to close the local socket and release the resources allocated for the client.

This event is only generated if the client is in non-blocking mode.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.OnError Event

Occurs when an client operation fails.

[Visual Basic]

```
Public Event OnError As OnErrorEventHandler
```

[C#]

```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type SmtpClient.ErrorEventArgs containing data related to this event. The following **SmtpClient.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a client operation fails.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see SmtpClient.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.SmtpClient.ErrorEventArgs

[Visual Basic]

```
Public Class SmtpClient.ErrorEventArgs
    Inherits EventArgs
```

[C#]

```
public class SmtpClient.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SmtpClient (in SocketTools.SmtpClient.dll)

See Also

SmtpClient.ErrorEventArgs Members | SocketTools Namespace

SmtpClient.ErrorEventArgs Members

SmtpClient.ErrorEventArgs overview

Public Instance Constructors

SmtpClient.ErrorEventArgs Constructor	Initializes a new instance of the SmtpClient.ErrorEventArgs class.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

SmtpClient.ErrorEventArgs Class | SocketTools Namespace

SmtpClient.ErrorEventArgs Constructor

Initializes a new instance of the SmtpClient.ErrorEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public SmtpClient.ErrorEventArgs();
```

See Also

SmtpClient.ErrorEventArgs Class | SocketTools Namespace

SmtpClient.ErrorEventArgs Properties

The properties of the **SmtpClient.ErrorEventArgs** class are listed below. For a complete list of **SmtpClient.ErrorEventArgs** class members, see the SmtpClient.ErrorEventArgs Members topic.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

SmtpClient.ErrorEventArgs Class | SocketTools Namespace

SmtpClient.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

SmtpClient.ErrorEventArgs Class | SocketTools Namespace | Error Property

SmtpClient.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public SmtpClient.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

SmtpClient.ErrorEventArgs Class | SocketTools Namespace | Description Property

SmtpClient.OnProgress Event

Occurs as a data stream is being read or written to the client.

[Visual Basic]

```
Public Event OnProgress As OnProgressEventHandler
```

[C#]

```
public event OnProgressEventHandler OnProgress;
```

Event Data

The event handler receives an argument of type SmtpClient.ProgressEventArgs containing data related to this event. The following **SmtpClient.ProgressEventArgs** properties provide information specific to this event.

Property	Description
BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
BytesTotal	Gets a value which specifies the total number of bytes in the data stream.
Percent	Gets a value which specifies the percentage of data that has been read or written.

Remarks

The **OnProgress** event occurs as a data stream is being read or written to the client. If large amounts of data are being read or written, this event can be used to update a progress bar or other user-interface component to provide the user with some visual feedback on the progress of the operation.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.ProgressEventArgs Class

Provides data for the OnProgress event.

For a list of all members of this type, see SmtpClient.ProgressEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.SmtpClient.ProgressEventArgs

[Visual Basic]
```
Public Class SmtpClient.ProgressEventArgs
    Inherits EventArgs
```

[C#]
```
public class SmtpClient.ProgressEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ProgressEventArgs specifies the number of bytes copied from the data stream, the total number of bytes in the data stream and a completion percentage.

The OnProgress event occurs as a data stream is being read or written to the client.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SmtpClient (in SocketTools.SmtpClient.dll)

See Also

SmtpClient.ProgressEventArgs Members | SocketTools Namespace

SmtpClient.ProgressEventArgs Members

SmtpClient.ProgressEventArgs overview

Public Instance Constructors

SmtpClient.ProgressEventArgs Constructor	Initializes a new instance of the SmtpClient.ProgressEventArgs class.

Public Instance Properties

BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
BytesTotal	Gets a value which specifies the total number of bytes in the data stream.
Percent	Gets a value which specifies the percentage of data that has been read or written.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

SmtpClient.ProgressEventArgs Class | SocketTools Namespace

SmtpClient.ProgressEventArgs Constructor

Initializes a new instance of the SmtpClient.ProgressEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public SmtpClient.ProgressEventArgs();
```

See Also

SmtpClient.ProgressEventArgs Class | SocketTools Namespace

SmtpClient.ProgressEventArgs Properties

The properties of the **SmtpClient.ProgressEventArgs** class are listed below. For a complete list of **SmtpClient.ProgressEventArgs** class members, see the SmtpClient.ProgressEventArgs Members topic.

Public Instance Properties

BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
BytesTotal	Gets a value which specifies the total number of bytes in the data stream.
Percent	Gets a value which specifies the percentage of data that has been read or written.

See Also

SmtpClient.ProgressEventArgs Class | SocketTools Namespace

SmtpClient.ProgressEventArgs.BytesCopied Property

Gets a value which specifies the number of bytes of data that has been read or written.

[Visual Basic]

```
Public ReadOnly Property BytesCopied As Integer
```

[C#]

```
public int BytesCopied {get;}
```

Property Value

An integer value which specifies the number of bytes of data.

Remarks

The **BytesCopied** property specifies the number of bytes that have been read from the client and stored in the local stream buffer, or written from the stream buffer to the client.

See Also

SmtpClient.ProgressEventArgs Class | SocketTools Namespace | BytesTotal Property | Percent Property

SmtpClient.ProgressEventArgs.BytesTotal Property

Gets a value which specifies the total number of bytes in the data stream.

[Visual Basic]
```
Public ReadOnly Property BytesTotal As Integer
```

[C#]
```
public int BytesTotal {get;}
```

Property Value

An integer value which specifies the number of bytes of data.

Remarks

The **BytesTotal** property specifies the total amount of data being read from the client and stored in the data stream, or written from the data stream to the client. If the amount of data was unknown or unspecified at the time the method call was made, then this value will always be the same as the **BytesCopied** property.

See Also

SmtpClient.ProgressEventArgs Class | SocketTools Namespace | BytesCopied Property | Percent Property

SmtpClient.ProgressEventArgs.Percent Property

Gets a value which specifies the percentage of data that has been read or written.

[Visual Basic]

```
Public ReadOnly Property Percent As Integer
```

[C#]

```
public int Percent {get;}
```

Property Value

An integer value which specifies a percentage.

Remarks

The **Percent** property specifies the percentage of data that has been transmitted, expressed as an integer value between 0 and 100, inclusive. If the maximum size of the data stream was not specified by the caller, this value will always be 100.

See Also

SmtpClient.ProgressEventArgs Class | SocketTools Namespace | BytesCopied Property | BytesTotal Property

SmtpClient.OnTimeout Event

Occurs when a blocking operation fails to complete before the timeout period elapses.

[Visual Basic]

```
Public Event OnTimeout As EventHandler
```

[C#]

```
public event EventHandler OnTimeout;
```

Remarks

The **OnTimeout** event occurs when a blocking operation, such as sending or receiving data on the client, fails to complete before the specified timeout period elapses. The timeout period for a blocking operation can be adjusted by setting the **Timeout** property.

This event is only generated if the client is in blocking mode.

See Also

SmtpClient Class | SocketTools Namespace

SmtpClient.ErrorCode Enumeration

Specifies the error codes returned by the SmtpClient class.

[Visual Basic]

```
Public Enum SmtpClient.ErrorCode
```

[C#]

```
public enum SmtpClient.ErrorCode
```

Remarks

The SmtpClient class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.
errorProxyAuthenticationRequired	Proxy authentication required.

errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.
errorUnsupportedMediaType	Request specified an unsupported media type.

errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.
errorNoMoreMailboxes	No more mailboxes exist on this server.

errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.
errorInvalidThreadHandle	The specified thread handle is no longer valid.

errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been defined.

errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or unrecognized.

errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SmtpClient (in SocketTools.SmtpClient.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

SmtpClient.SecureCipherAlgorithm Enumeration

Specifies the encryption algorithms that the SmtpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum SmtpClient.SecureCipherAlgorithm
```

[C#]

```
[Flags]
public enum SmtpClient.SecureCipherAlgorithm
```

Remarks

The SmtpClient class uses the **SecureCipherAlgorithm** enumeration to identify which encryption algorithm was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
cipherNone	No cipher has been selected. A secure connection has not been established with the remote host.	0
cipherRC2	The RC2 block cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.	1
cipherRC4	The RC4 stream cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.	2
cipherRC5	The RC5 block cipher was selected. This is a variable key length cipher which supports keys up to 2040 bits, in 8-bit increments.	4
cipherDES	The DES (Data Encryption Standard) block cipher was selected. This is a fixed key length cipher using 56-bit keys.	8
cipherDES3	The Triple DES block cipher was selected. This cipher encrypts the data three times using different keys, effectively using a 168-bit key length.	16
cipherDESX	A variant of the DES block cipher which XORs an extra 64-bits of the key before and after the plaintext has been encrypted, increasing the key size to 184 bits.	32
cipherAES	The Advanced Encryption Standard	64

	cipher (also known as the Rijndael cipher) is a fixed block size cipher which use a key size of 128, 192 or 256 bits. This cipher is supported on Windows XP SP3 SP3 and later versions of the operating system.	
cipherSkipjack	The Skipjack block cipher was selected. This is a fixed key length cipher, using 80-bit keys.	128

Requirements

Namespace: SocketTools

Assembly: SocketTools.SmtpClient (in SocketTools.SmtpClient.dll)

See Also

SocketTools Namespace

SmtpClient.SecureHashAlgorithm Enumeration

Specifies the hash algorithms that the SmtpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum SmtpClient.SecureHashAlgorithm
```

```
[C#]
[Flags]
public enum SmtpClient.SecureHashAlgorithm
```

Remarks

The SmtpClient class uses the **SecureHashAlgorithm** enumeration to identify the message digest (hash) algorithm that was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
hashNone	No hash algorithm has been selected. This is not a secure connection with the server.	0
hashMD5	The MD5 algorithm was selected. This algorithm produces a 128-bit message digest. This algorithm is no longer considered to be cryptographically secure.	1
hashSHA	The SHA-1 algorithm was selected. This algorithm produces a 160-bit message digest. This algorithm is no longer considered to be cryptographically secure.	2
hashSHA256	The SHA-256 algorithm was selected. This algorithm produces a 256-bit message digest.	4
hashSHA384	The SHA-384 algorithm was selected. This algorithm produces a 384-bit message digest.	8
hashSHA512	The SHA-512 algorithm was selected. This algorithm produces a 512-bit message digest.	16

Requirements

Namespace: SocketTools

Assembly: SocketTools.SmtpClient (in SocketTools.SmtpClient.dll)

See Also

SocketTools Namespace

SmtpClient.SecureKeyAlgorithm Enumeration

Specifies the key exchange algorithms that the SmtpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum SmtpClient.SecureKeyAlgorithm
```

[C#]
```
[Flags]
public enum SmtpClient.SecureKeyAlgorithm
```

Remarks

The SmtpClient class uses the **SecureKeyAlgorithm** enumeration to identify the key exchange algorithm that was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
keyExchangeNone	No key exchange algorithm has been selected. This is not a secure connection with the server.	0
keyExchangeRSA	The RSA public key exchange algorithm has been selected.	1
keyExchangeKEA	The KEA public key exchange algorithm has been selected. This is an improved version of the Diffie-Hellman public key algorithm.	2
keyExchangeDH	The Diffie-Hellman public key exchange algorithm has been selected.	4
keyExchangeECDH	The Elliptic Curve Diffie-Hellman key exchange algorithm was selected. This is a variant of the Diffie-Hellman algorithm which uses elliptic curve cryptography. This key exchange algorithm is only supported on Windows XP SP3 SP3 and later versions of the operating system.	8

Requirements

Namespace: SocketTools

Assembly: SocketTools.SmtpClient (in SocketTools.SmtpClient.dll)

See Also

SocketTools Namespace

SmtpClient.SecurityCertificate Enumeration

Specifies the security certificate status values that may be returned by the SmtpClient class.

[Visual Basic]

```
Public Enum SmtpClient.SecurityCertificate
```

[C#]

```
public enum SmtpClient.SecurityCertificate
```

Remarks

The SmtpClient class uses the **SecurityCertificate** enumeration to identify the current status of the certificate that was provided by the remote host when a secure connection was established.

Members

Member Name	Description
certificateNone	No certificate information is available. A secure connection was not established with the server.
certificateValid	The certificate is valid.
certificateNoMatch	The certificate is valid, however the domain name specified in the certificate does not match the domain name of the remote host. The application can examine the **CertificateSubject** property to determine the site the certificate was issued to.
certificateExpired	The certificate has expired and is no longer valid. The application can examine the **CertificateExpires** property to determine when the certificate expired.
certificateRevoked	The certificate has been revoked and is no longer valid. It is recommended that the application immediately terminate the connection if this status is returned.
certificateUntrusted	The certificate has not been issued by a trusted authority, or the certificate is not trusted on the local host. It is recommended that the application immediately terminate the connection if this status is returned.
certificateInvalid	The certificate is invalid. This typically indicates that the internal structure of the certificate is damaged. It is recommended that the application immediately terminate the connection if this status is returned.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SmtpClient (in SocketTools.SmtpClient.dll)

See Also

SocketTools Namespace

SmtpClient.SecurityProtocols Enumeration

Specifies the security protocols that the SmtpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum SmtpClient.SecurityProtocols
```

[C#]
```
[Flags]
public enum SmtpClient.SecurityProtocols
```

Remarks

The SmtpClient class uses the **SecurityProtocols** enumeration to specify one or more security protocols to be used when establishing a connection with a remote host. Multiple protocols may be specified if necessary and the actual protocol used will be negotiated with the remote host. It is recommended that most applications use **protocolDefault** when creating a secure connection.

Members

Member Name	Description	Value
protocolNone	No security protocol will be used, a secure connection will not be established.	0
protocolSSL2	The SSL 2.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections.	1
protocolSSL3	The SSL 3.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections. In most cases, this protocol is only selected if TLS is not supported by the server.	2
protocolTLS10	The TLS 1.0 protocol has been selected. This version of the protocol is commonly used by older servers and is the only version of TLS supported on Windows platforms prior to Windows 7 SP1 and Windows Server 2008 R2.	4
protocolTLS11	The TLS 1.1 protocol has been selected. This version of TLS is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of the	8

	operating system.	
protocolTLS12	The TLS 1.2 protocol has been selected. This is the default version of the protocol and is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of Windows. It is recommended that you use this version of TLS.	16
protocolSSL	Any version of the Secure Sockets Layer (SSL) protocol should be used. The actual protocol version used will be negotiated with the remote host.	3
protocolTLS	Any version of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with with preference for TLS 1.2.	28
protocolTLS1	Version 1.0, 1.1 or 1.2 of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with preference for TLS 1.2.	28
protocolDefault	The default selection of security protocols will be used when establishing a connection. The TLS 1.2, 1.1 and 1.0 protocols will be negotiated with the host, in that order of preference. This option will always request the latest version of the preferred security protocols and is the recommended value.	16
protocolUnknown	An unknown or unsupported security protocol has been specified. This value indicates an error condition.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.SmtpClient (in SocketTools.SmtpClient.dll)

See Also

SocketTools Namespace

SmtpClient.SmtpOptions Enumeration

Specifies the options that the SmtpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum SmtpClient.SmtpOptions
```

[C#]

```
[Flags]
public enum SmtpClient.SmtpOptions
```

Remarks

The SmtpClient class uses the **SmtpOptions** enumeration to specify one or more options to be used when establishing a connection with a remote host. Multiple options may be specified if necessary.

Members

Member Name	Description	Value
optionNone	No option specified.	0
optionDefault	The default connection option. This is the same as specifying **optionExtended**.	1
optionExtended	Extended SMTP commands should be used if possible. This option enables features such as authentication and delivery status notification. If this option is not specified, the class will not attempt to use any extended features.	1
optionTunnel	This option specifies that a tunneled TCP connection and/or port-forwarding is being used to establish the connection to the server. This changes the behavior of the client with regards to internal checks of the destination IP address and remote port number, default capability selection and how the connection is established. This option also forces all connections to be outbound and enables the firewall compatibility features in the client.	1024
optionTrustedSite	This option specifies the server is trusted. The server certificate will not be validated and the connection will always be permitted. This option only affects connections using either the SSL or TLS protocols.	2048
optionSecure	This option specifies the client should	4096

	attempt to establish a secure connection with the server. The server must support secure connections using either the SSL or TLS protocol.	
optionExplicitSSL	This option specifies the client should attempt to establish a secure explicit SSL session. The initial connection to the server is not encrypted, and the client will attempt to negotiate a secure connection by sending the STARTTLS command to the server. Some servers may require this option when connecting to the server on ports other than the default secure port of 465.	4096
optionImplicitSSL	This option specifies the client should attempt to establish a secure implicit SSL session. The SSL handshake is initiated immediately after the connection to the server has been established.	8192
optionSecureFallback	This option specifies the client should permit the use of less secure cipher suites for compatibility with legacy servers. If this option is specified, the client will permit connections using TLS 1.0 and cipher suites that use RC4, MD5 and SHA1.	32768
optionPreferIPv6	This option specifies the client should prefer the use of IPv6 if the server hostname can be resolved to both an IPv6 and IPv4 address. This option is ignored if the local system does not have IPv6 enabled, or when the hostname can only be resolved to an IPv4 address. If the server hostname can only be resolved to an IPv6 address, the client will attempt to establish a connection using IPv6 regardless if this option has been specified.	262144
optionFreeThread	This option specifies that class methods may be called from any thread, and not only the thread that established the connection. Using this option disables certain internal safety checks that are made by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads.	524288

Requirements

Namespace: SocketTools

Assembly: SocketTools.SmtpClient (in SocketTools.SmtpClient.dll)

See Also

SocketTools Namespace

SmtpClient.SmtpStatus Enumeration

Specifies the status values that may be returned by the SmtpClient class.

[Visual Basic]
```
Public Enum SmtpClient.SmtpStatus
```

[C#]
```
public enum SmtpClient.SmtpStatus
```

Remarks

The SmtpClient class uses the **SmtpStatus** enumeration to identify the current status of the client.

Members

Member Name	Description
statusUnused	A client session has not been created. Attempts to perform any network operations, such as sending or receiving data, will generate an error.
statusIdle	A client session has been created, but is not currently in use. A blocking socket operation can be executed at this point.
statusConnect	The client is in the process of establishing a connection with a remote host.
statusRead	The client is in the process of receiving data from a remote host.
statusWrite	The client is in the process of sending data to a remote host.
statusDisconnect	The client session is being closed and subsequent attempts to access the client will result in an error.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SmtpClient (in SocketTools.SmtpClient.dll)

See Also

SocketTools Namespace

SmtpClient.TraceOptions Enumeration

Specifies the logging options that the SmtpClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum SmtpClient.TraceOptions
```

```
[C#]
[Flags]
public enum SmtpClient.TraceOptions
```

Remarks

The SmtpClient class uses the **TraceOptions** enumeration to specify what kind of debugging information is written to the trace logfile. These options are only meaningful when trace logging is enabled by setting the **Trace** property to **true**.

Members

Member Name	Description	Value
traceDefault	The default trace logging option. This is the same as specifying the **traceInfo** option.	0
traceInfo	All network function calls are written to the trace file. This is the default value.	0
traceError	Only those network function calls which fail are recorded in the trace file.	1
traceWarning	Only those network function calls which fail, or return values which indicate a warning, are recorded in the trace file.	2
traceHexDump	All network function calls are written to the trace file, plus all the data that is sent or received is displayed, in both ASCII and hexadecimal format.	4
traceProcess	All function calls in the current process are logged, rather than only those functions in the current thread. This option is useful for multithreaded applications that are using worker threads.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.SmtpClient (in SocketTools.SmtpClient.dll)

See Also

SocketTools Namespace

SmtpClient.OnCommandEventHandler Delegate

Represents the method that will handle the OnCommand event.

[Visual Basic]

```
Public Delegate Sub SmtpClient.OnCommandEventHandler( _
   ByVal sender As Object, _
   ByVal e As CommandEventArgs _
)
```

[C#]

```
public delegate void SmtpClient.OnCommandEventHandler(
      object sender,
      CommandEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A CommandEventArgs object that contains the event data.

Remarks

When you create an **OnCommandEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnCommandEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SmtpClient (in SocketTools.SmtpClient.dll)

See Also

SocketTools Namespace

SmtpClient.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

```
[Visual Basic]
Public Delegate Sub SmtpClient.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

```
[C#]
public delegate void SmtpClient.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender
The source of the event.

e
An ErrorEventArgs that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SmtpClient (in SocketTools.SmtpClient.dll)

See Also

SocketTools Namespace

SmtpClient.OnProgressEventHandler Delegate

Represents the method that will handle the OnProgress event.

[Visual Basic]

```
Public Delegate Sub SmtpClient.OnProgressEventHandler( _
   ByVal sender As Object, _
   ByVal e As ProgressEventArgs _
)
```

[C#]

```
public delegate void SmtpClient.OnProgressEventHandler(
      object sender,
      ProgressEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A ProgressEventArgs that contains the event data.

Remarks

When you create an **OnProgressEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnProgressEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SmtpClient (in SocketTools.SmtpClient.dll)

See Also

SocketTools Namespace

SmtpClient.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see SmtpClient.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.SmtpClient.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class SmtpClient.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class SmtpClient.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the SmtpClient class.

Example

```
<Assembly: SocketTools.SmtpClient.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.SmtpClient.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.SmtpClient (in SocketTools.SmtpClient.dll)

See Also

SmtpClient.RuntimeLicenseAttribute Members | SocketTools Namespace

SmtpClient.RuntimeLicenseAttribute Members

SmtpClient.RuntimeLicenseAttribute overview

Public Instance Constructors

SmtpClient.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

SmtpClient.RuntimeLicenseAttribute Class | SocketTools Namespace

SmtpClient.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public SmtpClient.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the SmtpClient class.

See Also

SmtpClient.RuntimeLicenseAttribute Class | SocketTools Namespace

SmtpClient.RuntimeLicenseAttribute Properties

The properties of the **SmtpClient.RuntimeLicenseAttribute** class are listed below. For a complete list of **SmtpClient.RuntimeLicenseAttribute** class members, see the SmtpClient.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

SmtpClient.RuntimeLicenseAttribute Class | SocketTools Namespace

SmtpClient.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

SmtpClient.RuntimeLicenseAttribute Class | SocketTools Namespace

SmtpClientException Class

The exception that is thrown when a client error occurs.

For a list of all members of this type, see SmtpClientException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.SmtpClientException

[Visual Basic]

```
Public Class SmtpClientException
    Inherits ApplicationException
```

[C#]

```
public class SmtpClientException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A SmtpClientException is thrown by the SmtpClient class when an error occurs.

The default constructor for the SmtpClientException class sets the **ErrorCode** property to the last client error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SmtpClient (in SocketTools.SmtpClient.dll)

See Also

SmtpClientException Members | SocketTools Namespace

SmtpClientException Members

SmtpClientException overview

Public Instance Constructors

SmtpClientException	Overloaded. Initializes a new instance of the SmtpClientException class.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

SmtpClientException Class | SocketTools Namespace

SmtpClientException Constructor

Initializes a new instance of the SmtpClientException class with the last network error code.

Overload List

Initializes a new instance of the SmtpClientException class with the last network error code.

public SmtpClientException();

Initializes a new instance of the SmtpClientException class with a specified error number.

public SmtpClientException(int);

Initializes a new instance of the SmtpClientException class with a specified error message.

public SmtpClientException(string);

Initializes a new instance of the SmtpClientException class with a specified error message and a reference to the inner exception that is the cause of this exception.

public SmtpClientException(string,Exception);

See Also

SmtpClientException Class | SocketTools Namespace

SmtpClientException Constructor ()

Initializes a new instance of the SmtpClientException class with the last network error code.

[Visual Basic]

```
Overloads Public Sub New()
```

[C#]

```
public SmtpClientException();
```

Remarks

The ctor constructor sets the **ErrorCode** property to the last client error that occurred. For more information about the errors that may occur, refer to the SmtpClient.ErrorCode enumeration.

See Also

SmtpClientException Class | SocketTools Namespace | SmtpClientException Constructor Overload List

SmtpClientException Constructor (String)

Initializes a new instance of the SmtpClientException class with a specified error message.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]

```
public SmtpClientException(
   string message
);
```

Parameters

message

The error message that explains the reason for the exception.

Remarks

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

SmtpClientException Class | SocketTools Namespace | SmtpClientException Constructor Overload List

SmtpClientException Constructor (String, Exception)

Initializes a new instance of the SmtpClientException class with a specified error message and a reference to the inner exception that is the cause of this exception.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal innerException As Exception _
)
```

[C#]

```
public SmtpClientException(
   string message,
   Exception innerException
);
```

Parameters

message
The error message that explains the reason for the exception.

innerException
The exception that is the cause of the current exception. If the ***innerException*** parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

SmtpClientException Class | SocketTools Namespace | SmtpClientException Constructor Overload List

SmtpClientException Constructor (Int32)

Initializes a new instance of the SmtpClientException class with a specified error number.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

[C#]

```
public SmtpClientException(
   int code
);
```

Parameters

code

An integer value which specifies an error code.

Remarks

This constructor sets the **ErrorCode** property to the specified error code. For more information about the errors that may occur, refer to the SmtpClient.ErrorCode enumeration.

See Also

SmtpClientException Class | SocketTools Namespace | SmtpClientException Constructor Overload List

SmtpClientException Properties

The properties of the **SmtpClientException** class are listed below. For a complete list of **SmtpClientException** class members, see the SmtpClientException Members topic.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

SmtpClientException Class | SocketTools Namespace

SmtpClientException.ErrorCode Property

Gets a value which specifies the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property ErrorCode As ErrorCode
```

[C#]

```
public SmtpClient.ErrorCode ErrorCode {get;}
```

Property Value

Returns a SmtpClient.ErrorCode enumeration value which specifies the error code.

Remarks

The **ErrorCode** property returns the error code that specifies the cause of the exception.

The default constructor for the SmtpClientException class sets the error code to the last network error that occurred. For more information about the errors that may occur, refer to the SmtpClient.ErrorCode enumeration.

See Also

SmtpClientException Class | SocketTools Namespace

SmtpClientException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]

```
Overrides Public ReadOnly Property Message As String
```

[C#]

```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

SmtpClientException Class | SocketTools Namespace

SmtpClientException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception. This value is the same as the values returned by the Windows Sockets API. For more information about socket error codes, see the Windows Socket Version 2 API error code documentation in MSDN.

See Also

SmtpClientException Class | SocketTools Namespace

SmtpClientException Methods

The methods of the **SmtpClientException** class are listed below. For a complete list of **SmtpClientException** class members, see the SmtpClientException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

SmtpClientException Class | SocketTools Namespace

SmtpClientException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]

```
Overrides Public Function ToString() As String
```

[C#]

```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

SmtpClientException Class | SocketTools Namespace

SocketWrench Class

A general purpose TCP/IP networking class for developing client and server applications.

For a list of all members of this type, see SocketWrench Members.

System.Object
  SocketTools.SocketWrench

[Visual Basic]

```
Public Class SocketWrench
    Implements IDisposable
```

[C#]

```
public class SocketWrench : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

At the core of each of the SocketTools networking classes is the Windows Sockets API. This provides a low level interface for sending and receiving data over the Internet or a local intranet using the Transmission Control Protocol (TCP) and/or User Datagram Protocol (UDP). The SocketWrench class provides a simpler interface to the Windows Sockets API, without sacrificing features or functionality. Using SocketWrench, you can easily create client and server applications while avoiding many of the mundane tasks and common problems that programmers face when developing Internet applications.

This class supports secure connections using the standard SSL and TLS protocols and can also be used to create secure, custom server programs. Both implicit and explicit SSL connections are supported, enabling the class to work with a wide variety of client and server applications without requiring that you use third-party classes or understand Microsoft's cryptography classes.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SocketWrench (in SocketTools.SocketWrench.dll)

See Also

SocketWrench Members | SocketTools Namespace

SocketWrench Members

SocketWrench overview

Public Instance Constructors

SocketWrench Constructor	Initializes a new instance of the SocketWrench class.

Public Instance Fields

AdapterAddress	Returns the IP address associated with the specified network adapter.
HostAlias	Returns the aliases for a given host name.

Public Instance Properties

AdapterCount	Get the number of available local and remote network adapters.
AddressFamily	Gets and sets a value that determines which version of the Internet Protocol will be used.
AtMark	Get a value that indicates if the next receive will return urgent data.
AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
Backlog	Gets and sets a value that indicates the number of connections that may be queued for a listening socket.
Blocking	Gets and sets a value which indicates if the socket is in blocking mode.
Broadcast	Gets and sets a value which indicates if datagrams will be broadcast over the local network.
ByteOrder	Gets and sets a value which indicates how integer data is read and written to the socket.
CertificateExpires	Get a value that specifies the date that the security certificate expires.
CertificateIssued	Get a value that specifies the date that the security certificate was issued.
CertificateIssuer	Get a value that provides information about the organization that issued the certificate.
CertificateName	Gets and sets a value that specifies the name of the security certificate.
CertificatePassword	Gets and sets the password associated with the security certificate.
CertificateStatus	Gets a value which indicates the status of the security certificate returned by the remote host.

CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateSubject	Gets a value that provides information about the organization that the server certificate was issued to.
CertificateUser	Gets and sets the user that owns the security certificate.
CipherStrength	Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.
CodePage	Gets and sets the code page used when reading and writing text.
ExternalAddress	Gets a value that specifies the external Internet address for the local system.
Handle	Gets a value that specifies the socket handle allocated for the current session.
HashStrength	Gets a value which specifies the length of the message digest that was selected for a secure connection.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostFile	Gets and sets a value that specifies the name of a host file used to resolve host names and addresses.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
InLine	Gets and sets a value that indicates if urgent data is received in-line with non-urgent data.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking socket operation.
IsClosed	Gets a value which indicates if the connection to the remote host has been closed.
IsConnected	Gets a value which indicates if a connection to the remote host has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsListening	Gets a value which indicates if the socket is listening for client connections.
IsReadable	Gets a value which indicates if there is data available to be read from the socket.
IsWritable	Gets a value which indicates if data can be written to the socket without blocking.
KeepAlive	Gets and sets a value which indicates if keep-alive

	packets are sent on a connected socket.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
Linger	Gets and sets a value which specifies the number of seconds to wait for the socket to disconnect from the remote host.
LocalAddress	Gets and sets the local Internet address that the socket will be bound to.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets and sets a value which specifies the local port number the socket will be bound to.
LocalService	Gets and sets a value which specifies the local service the socket will be bound to.
NoDelay	Gets and sets a value which specifies if the Nagle algorithm should be enabled or disabled.
Options	Gets and sets a value which specifies one or more socket options.
PeerAddress	Gets a value that specifies the Internet address of the remote host.
PeerName	Gets a value that specifies the name of the remote host.
PeerPort	Gets a value that specifies the port number used by the remote host.
PhysicalAddress	Gets a value which specifies the MAC address for the local system's network adapter.
Protocol	Gets and sets a value which specifies the socket protocol.
RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.
ReservedPort	Gets and sets a value which indicates if a reserved port number was used.
ReuseAddress	Gets and sets a value which indicates if a socket address can be reused.
Route	Gets and sets a value which indicates if packets should be routed.
Secure	Gets and sets a value which specifies if a secure connection is established.

SecureCipher	Gets a value that specifies the encryption algorithm used for a secure connection.
SecureHash	Gets a value that specifies the message digest algorithm used for a secure connection.
SecureKeyExchange	Gets a value that specifies the key exchange algorithm used for a secure connection.
SecureProtocol	Gets and sets a value which specifies the protocol used for a secure connection.
Status	Gets a value which specifies the current status of the socket.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the network function tracing logfile.
TraceFlags	Gets and sets a value which specifies the network function tracing flags.
Urgent	Gets and sets a value which specifies if urgent data will be read or written.
Version	Gets a value which returns the current version of the SocketWrench class library.

Public Instance Methods

Abort	Abort the connection with a remote host.
Accept	Overloaded. Accepts a client connection on a listening socket, specifying a timeout period and one or more socket options.
AttachThread	Attach an instance of the class to the current thread
Bind	Overloaded. Bind the socket to the specified local address and port number.
Cancel	Cancel the current blocking socket operation.
Connect	Overloaded. Establish a connection with a remote host.
Disconnect	Terminate the connection with a remote host.
Dispose	Overloaded. Releases all resources used by SocketWrench.

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the SocketWrench class.
Listen	Overloaded. Listen for incoming client connections, specifying the local network address, port number and connection backlog.
Peek	Overloaded. Read data from the socket and store it in a byte array, but do not remove the data from the socket buffers.
Read	Overloaded. Read data from the socket and store it in a byte array.
ReadFrom	Overloaded. Read data from the socket and store it in a byte array.
ReadLine	Overloaded. Read up to a line of data from the socket and return it in a string buffer.
ReadStream	Overloaded. Read a data stream from the socket and store it in the specified byte array.
Reject	Rejects a connection request from a remote host.
Reset	Reset the internal state of the object, resetting all properties to their default values.
Resolve	Resolves a host name to a host IP address.
Shutdown	Overloaded. Disable sending or receiving data on the socket.
StoreStream	Overloaded. Reads a data stream from the socket and stores it in the specified file.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
Write	Overloaded. Write one or more bytes of data to the socket.
WriteLine	Overloaded. Send a line of text to the remote host, terminated by a carriage-return and linefeed.
WriteStream	Overloaded. Write a stream of bytes to the socket.
WriteTo	Overloaded. Write one or more bytes of data to the socket.

Public Instance Events

OnAccept	Occurs when a remote host attempts to establish a connection with the local system.
OnCancel	Occurs when a blocking socket operation is canceled.
OnConnect	Occurs when a connection is established with the remote host.
OnDisconnect	Occurs when the remote host disconnects from the local system.
OnError	Occurs when an socket operation fails.
OnProgress	Occurs as a data stream is being read or written to the socket.
OnRead	Occurs when data is available to be read from the socket.
OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
OnWrite	Occurs when data can be written to the socket.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the SocketWrench class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench Constructor

Initializes a new instance of the SocketWrench class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public SocketWrench();
```

Example

The following example demonstrates creating an instance of the **SocketWrench** class object and resolving a hostname into an Internet address using the Resolve method.

```
Dim Socket As SocketTools.SocketWrench
Dim strHostName As String
Dim strHostAddress As String

Socket = New SocketTools.SocketWrench
strHostName = TextBox1.Text.Trim()

If Socket.Resolve(strHostName, strHostAddress) Then
    StatusBar1.Text = "The Internet address for " + strHostName + " is " +
strHostAddress
Else
    StatusBar1.Text = "The Internet address for " + strHostName + " could not be
resolved"
End If
```

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench Fields

The fields of the **SocketWrench** class are listed below. For a complete list of **SocketWrench** class members, see the SocketWrench Members topic.

Public Instance Fields

AdapterAddress	Returns the IP address associated with the specified network adapter.
HostAlias	Returns the aliases for a given host name.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.AdapterAddress Field

Returns the IP address associated with the specified network adapter.

[Visual Basic]

```
Public ReadOnly AdapterAddress As AdapterAddressArray
```

[C#]

```
public readonly AdapterAddressArray AdapterAddress;
```

Remarks

The **AdapterAddress** array returns the IP addresses that are associated with the local network or remote dial-up network adapters configured on the system. The **AdapterCount** property can be used to determine the number of adapters that are available.

Multihomed systems with more than one local network adapter, or a combination of local and dial-up adapters will not be listed in a specific order. An application should not make the assumption that the first address returned by **AdapterAddress** always refers to a local network adapter.

Note that it is possible that the **AdapterCount** property will return 0, and **AdapterAddress** will return an empty string. This indicates that the system does not have a physical network adapter with an assigned IP address, and there are no dial-up networking connections currently active. If a dial-up networking connection is established at some later point, the **AdapterCount** property will change to 1, and the **AdapterAddress** property will return the IP address allocated for that connection.

See Also

SocketWrench Class | SocketTools Namespace | AdapterAddressArray Class | AdapterCount Property

SocketWrench.HostAlias Field

Returns the aliases for a given host name.

[Visual Basic]

```
Public ReadOnly HostAlias As HostAliasArray
```

[C#]

```
public readonly HostAliasArray HostAlias;
```

Remarks

The **HostAlias** array returns the aliases assigned to the host specified by the **HostAddress** or **HostName** properties. If the host address or name can be resolved, the first element in the **HostAlias** array always refers to the host's fully qualified domain name.

The end of the alias list is indicated when the property returns an empty string. The array is zero based, meaning that the first index value is zero.

Example

```
Dim nIndex As Integer

ListBox1.Items.Clear()
Socket.HostName = strHostName

For nIndex = 0 To Socket.HostAliases - 1
    ListBox1.Items.Add(Socket.HostAlias(nIndex))
Next
```

See Also

SocketWrench Class | SocketTools Namespace | HostAliasArray Class

SocketWrench Properties

The properties of the **SocketWrench** class are listed below. For a complete list of **SocketWrench** class members, see the SocketWrench Members topic.

Public Instance Properties

AdapterCount	Get the number of available local and remote network adapters.
AddressFamily	Gets and sets a value that determines which version of the Internet Protocol will be used.
AtMark	Get a value that indicates if the next receive will return urgent data.
AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
Backlog	Gets and sets a value that indicates the number of connections that may be queued for a listening socket.
Blocking	Gets and sets a value which indicates if the socket is in blocking mode.
Broadcast	Gets and sets a value which indicates if datagrams will be broadcast over the local network.
ByteOrder	Gets and sets a value which indicates how integer data is read and written to the socket.
CertificateExpires	Get a value that specifies the date that the security certificate expires.
CertificateIssued	Get a value that specifies the date that the security certificate was issued.
CertificateIssuer	Get a value that provides information about the organization that issued the certificate.
CertificateName	Gets and sets a value that specifies the name of the security certificate.
CertificatePassword	Gets and sets the password associated with the security certificate.
CertificateStatus	Gets a value which indicates the status of the security certificate returned by the remote host.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateSubject	Gets a value that provides information about the organization that the server certificate was issued to.
CertificateUser	Gets and sets the user that owns the security certificate.
CipherStrength	Gets a value that indicates the length of the key

	used by the encryption algorithm for a secure connection.
CodePage	Gets and sets the code page used when reading and writing text.
ExternalAddress	Gets a value that specifies the external Internet address for the local system.
Handle	Gets a value that specifies the socket handle allocated for the current session.
HashStrength	Gets a value which specifies the length of the message digest that was selected for a secure connection.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostFile	Gets and sets a value that specifies the name of a host file used to resolve host names and addresses.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
InLine	Gets and sets a value that indicates if urgent data is received in-line with non-urgent data.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking socket operation.
IsClosed	Gets a value which indicates if the connection to the remote host has been closed.
IsConnected	Gets a value which indicates if a connection to the remote host has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsListening	Gets a value which indicates if the socket is listening for client connections.
IsReadable	Gets a value which indicates if there is data available to be read from the socket.
IsWritable	Gets a value which indicates if data can be written to the socket without blocking.
KeepAlive	Gets and sets a value which indicates if keep-alive packets are sent on a connected socket.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
Linger	Gets and sets a value which specifies the number of seconds to wait for the socket to disconnect from the remote host.

LocalAddress	Gets and sets the local Internet address that the socket will be bound to.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets and sets a value which specifies the local port number the socket will be bound to.
LocalService	Gets and sets a value which specifies the local service the socket will be bound to.
NoDelay	Gets and sets a value which specifies if the Nagle algorithm should be enabled or disabled.
Options	Gets and sets a value which specifies one or more socket options.
PeerAddress	Gets a value that specifies the Internet address of the remote host.
PeerName	Gets a value that specifies the name of the remote host.
PeerPort	Gets a value that specifies the port number used by the remote host.
PhysicalAddress	Gets a value which specifies the MAC address for the local system's network adapter.
Protocol	Gets and sets a value which specifies the socket protocol.
RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.
ReservedPort	Gets and sets a value which indicates if a reserved port number was used.
ReuseAddress	Gets and sets a value which indicates if a socket address can be reused.
Route	Gets and sets a value which indicates if packets should be routed.
Secure	Gets and sets a value which specifies if a secure connection is established.
SecureCipher	Gets a value that specifies the encryption algorithm used for a secure connection.
SecureHash	Gets a value that specifies the message digest algorithm used for a secure connection.
SecureKeyExchange	Gets a value that specifies the key exchange algorithm used for a secure connection.
SecureProtocol	Gets and sets a value which specifies the protocol used for a secure connection.

Status	Gets a value which specifies the current status of the socket.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the network function tracing logfile.
TraceFlags	Gets and sets a value which specifies the network function tracing flags.
Urgent	Gets and sets a value which specifies if urgent data will be read or written.
Version	Gets a value which returns the current version of the SocketWrench class library.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.AdapterCount Property

Get the number of available local and remote network adapters.

[Visual Basic]

```
Public ReadOnly Property AdapterCount As Integer
```

[C#]

```
public int AdapterCount {get;}
```

Property Value

Returns the number of available local and remote network adapters.

Remarks

The **AdapterCount** property returns the number of local and remote dial-up networking adapters available on the local system. This value can be used in conjunction with the **AdapterAddress** array to enumerate the IP addresses assigned to the various network adapters.

Note that it is possible that the **AdapterCount** property will return 0, and **AdapterAddress** will return an empty string. This indicates that the system does not have a physical network adapter with an assigned IP address, and there are no dial-up networking connections currently active. If a dial-up networking connection is established at some later point, the **AdapterCount** property will change to 1, and the **AdapterAddress** property will return the IP address allocated for that connection.

See Also

SocketWrench Class | SocketTools Namespace | AdapterAddress Field

SocketWrench.AtMark Property

Get a value that indicates if the next receive will return urgent data.

[Visual Basic]

```
Public ReadOnly Property AtMark As Boolean
```

[C#]

```
public bool AtMark {get;}
```

Property Value

Returns **true** if the next read on the socket will return urgent data.

Remarks

This property can only be used if the **Protocol** property is set to **SocketProtocol.socketStream** and the **InLine** property has been set to **true**.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.AutoResolve Property

Gets and sets a value that determines if host names and addresses are automatically resolved.

[Visual Basic]

```
Public Property AutoResolve As Boolean
```

[C#]

```
public bool AutoResolve {get; set;}
```

Property Value

Returns **true** if host names are automatically resolved to Internet addresses. The default value is **false**.

Remarks

Setting the **AutoResolve** property determines if the class automatically resolves host names and addresses specified by the **HostName** and **HostAddress** properties. If set to **true**, setting the **HostName** property will cause the class to automatically determine the corresponding IP address and update the **HostAddress** property accordingly. Likewise, setting the **HostAddress** property will cause the class to determine the host name and update the **HostName** property. Setting this property to **false** prevents the class from resolving host names until a connection attempt is made.

It is important to note that setting the **HostName** or **HostAddress** property may cause the current thread to block, sometimes for several seconds, until the name or address is resolved. To prevent this behavior, set this property value to **false**.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Backlog Property

Gets and sets a value that indicates the number of connections that may be queued for a listening socket.

[Visual Basic]

```
Public Property Backlog As Integer
```

[C#]

```
public int Backlog {get; set;}
```

Property Value

Returns an integer value that specifies the size of the backlog queue. The default value is 5.

Remarks

The **Backlog** property specifies the maximum size of the queue used to manage pending connections to the service. If the property is set to value which exceeds the maximum size for the underlying service provider, it will be silently adjusted to the nearest legal value. There is no standard way to determine what the maximum backlog value is.

This property must be set to the desired value before the **Listen** method is called, if the **Listen** method is used with default parameters.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Blocking Property

Gets and sets a value which indicates if the socket is in blocking mode.

[Visual Basic]
```
Public Property Blocking As Boolean
```

[C#]
```
public bool Blocking {get; set;}
```

Property Value

Returns **true** if the socket is in blocking mode; otherwise it returns **false**. The default value is **true**.

Remarks

Setting the **Blocking** property determines if socket operations complete synchronously or asynchronously. If set to **true**, then each socket operation (such as sending or receiving data) will return when the operation has completed or timed-out. If set to **false**, socket operations will return immediately. If the operation would result in the socket blocking (such as attempting to read data when no data has been sent by the remote host), an error is generated.

It is important to note that certain events, such as **OnDisconnect, OnRead** and **OnWrite** are only fired if the socket is in non-blocking mode.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Broadcast Property

Gets and sets a value which indicates if datagrams will be broadcast over the local network.

[Visual Basic]
```
Public Property Broadcast As Boolean
```

[C#]
```
public bool Broadcast {get; set;}
```

Property Value

Returns **true** if datagrams will be broadcast; otherwise returns **false**. The default value is **false**.

Remarks

If the **Broadcast** property is set to a value of **true**, the datagram written to the socket will be broadcast to all systems on the network. Use of this property is restricted to the UDP protocol and the value is ignored for TCP connections.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.ByteOrder Property

Gets and sets a value which indicates how integer data is read and written to the socket.

[Visual Basic]

```
Public Property ByteOrder As SocketByteOrder
```

[C#]

```
public SocketWrench.SocketByteOrder ByteOrder {get; set;}
```

Property Value

A SocketByteOrder enumeration value which specifies the byte order. The default is **byteOrderNative**.

Remarks

The **ByteOrder** property is used to specify how integer data is written to and read from the socket. The default value for this property is **byteOrderNative**, which specifies that integers should be written in the native byte order for the local machine. A value of **byteOrderNetwork** indicates that integers should be written in network byte order.

When applications write integer values on a socket (instead of string representations of those values), they should typically be converted to network byte order before they are sent. Likewise, when an integer value is read, it should then be converted from the network byte order back to the byte order used by the local machine. The native byte order, also called the host byte order, should only be used if it can be assured that both the sender and the receiver are running on an identical or compatible machine architectures (for example, if both systems are Intel-based).

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.CertificateExpires Property

Get a value that specifies the date that the security certificate expires.

[Visual Basic]

```
Public ReadOnly Property CertificateExpires As String
```

[C#]

```
public string CertificateExpires {get;}
```

Property Value

A string which specifies a date using the local date and time format.

Remarks

The **CertificateExpires** property returns a string that specifies the date and time that the security certificate expires. This property will return an empty string if a secure connection has not been established with the remote host.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.CertificateIssued Property

Get a value that specifies the date that the security certificate was issued.

[Visual Basic]

```
Public ReadOnly Property CertificateIssued As String
```

[C#]

```
public string CertificateIssued {get;}
```

Property Value

A string which specifies a date using the local date and time format.

Remarks

The **CertificateIssued** property returns a string that specifies the date and time that the security certificate was issued. This property will return an empty string if a secure connection has not been established with the remote host.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.CertificateIssuer Property

Get a value that provides information about the organization that issued the certificate.

[Visual Basic]

```
Public ReadOnly Property CertificateIssuer As String
```

[C#]

```
public string CertificateIssuer {get;}
```

Property Value

A string that contains a comma separated list of name value pairs.

Remarks

The **CertificateIssuer** property returns a string that contains information about the organization that issued the server certificate. The string value is a comma separated list of tagged name and value pairs. In the nomenclature of the X.500 standard, each of these pairs are called a relative distinguished name (RDN), and when concatenated together, forms the issuer's distinguished name (DN). For example:

```
C=US, O="RSA Data Security, Inc.", OU=Secure Server Certification Authority
```

To obtain a specific value, such as the name of the issuer or the issuer's country, the application must parse the string returned by this property. Some of the common tokens used in the distinguished name are:

Token	Description
C	The ISO standard two character country code.
S	The name of the state or province.
L	The name of the city or locality.
O	The name of the company or organization.
OU	The name of the department or organizational unit
CN	The common name; with X.509 certificates, this is the domain name of the site the certificate was issued for.

This property will return an empty string if a secure connection has not been established with the remote host.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.CertificateName Property

Gets and sets a value that specifies the name of the security certificate.

[Visual Basic]
```
Public Property CertificateName As String
```

[C#]
```
public string CertificateName {get; set;}
```

Property Value

A string which specifies the certificate name.

Remarks

The **CertificateName** property sets the common name or friendly name of the certificate that should be used when establishing a secure client connection or accepting a secure connection from a remote host. This property is used in conjunction with the **CertificateStore** property to identify the client or server certificate.

For client applications, it is only required that you set this property value if the server requires a client certificate for authentication. If this property is not set, a client certificate will not be provided to the server. The certificate must be designated as a client certificate and have a private key associated with it, otherwise the connection attempt will fail.

For server applications, it is required that you specify a certificate name if security has been enabled by setting the **Secure** property to true. The certificate must be designated as a server certificate and have a private key associated with it, otherwise incoming client connections cannot be accepted.

When the certificate store is searched for a matching certificate, it will first search for any certificate with a friendly name that matches the property value. If no valid certificate is found, it will then search for a certificate with a matching common name.

Certificates may be installed and viewed on the local system using the Certificate Manager that is included with the Windows operating system. For more information, refer to the documentation for the Microsoft Management Console.

See Also

SocketWrench Class | SocketTools Namespace | CertificateStore Property | Secure Property

SocketWrench.CertificateStatus Property

Gets a value which indicates the status of the security certificate returned by the remote host.

[Visual Basic]

```
Public ReadOnly Property CertificateStatus As SecurityCertificate
```

[C#]

```
public SocketWrench.SecurityCertificate CertificateStatus {get;}
```

Property Value

A SecurityCertificate enumeration value which specifies the status of the certificate.

Remarks

The **CertificateStatus** property is used to determine the status of the security certificate returned by the remote host when a secure connection has been established. This property value should be checked after the connection to the server has completed, but prior to beginning a transaction.

Note that if the certificate cannot be validated, the secure connection will not be automatically terminated. It is the responsibility of your application to determine the best course of action to take if the certificate is invalid. Even if the security certificate cannot be validated, the data exchanged with the remote host will still be encrypted.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.CertificateStore Property

Gets and sets a value that specifies the name of the local certificate store.

```
[Visual Basic]
Public Property CertificateStore As String
```

```
[C#]
public string CertificateStore {get; set;}
```

Property Value

A string which specifies the certificate store name. The default value is the current user's personal certificate store.

Remarks

The **CertificateStore** property is used to specify the name of the certificate store which contains the security certificate to use when establishing a secure connection. The certificate may either be stored in the registry or in a file. If the certificate is stored in the registry, then this property should be set to one of the following predefined values:

Store Name	Description
CA	Certification authority certificates. These are certificates that are issued by entities which are entrusted to issue certificates to other individuals or organizations. Companies such as VeriSign and Thawte act as certification authorities.
MY	Personal certificates and their associated private keys for the current user. This store typically holds the client certificates used to establish a user's credentials. If a certificate store is not specified, this is the default value that is used.
ROOT	Certificates that have been self-signed by a certificate authority. Root certificates for a number of different certification authorities such as VeriSign and Thawte are installed as part of the operating system and periodically updated by Microsoft.

In most cases the client certificate will be installed in the user's personal certificate store, and therefore it is not necessary to set this property value because that is the default location that will be used to search for the certificate. This property is only used if the **CertificateName** property is also set to a valid certificate name.

If you are using a local certificate store, with the certificate and private key stored in the registry, you can explicitly specify whether the certificate store for the current user or the local machine (all users) should be used. This is done by prefixing the certificate store name with "HKCU" for the current user, or "HKLM" for the local machine. For example, a certificate store name of "HKLM:MY" would specify the personal certificate store for the local machine, rather than the current user. If neither prefix is specified, it will default to the certificate store for the current user.

This property may also be used to specify a file that contains the client certificate. In this case, the property should specify the full path to the file and must contain both the certificate and private key in PKCS12

format. If the file is protected by a password, the **CertificatePassword** property must also be set to specify the password.

See Also

SocketWrench Class | SocketTools Namespace | CertificatePassword Property | Secure Property

SocketWrench.CertificateSubject Property

Gets a value that provides information about the organization that the server certificate was issued to.

[Visual Basic]
```
Public ReadOnly Property CertificateSubject As String
```

[C#]
```
public string CertificateSubject {get;}
```

Property Value

A string that contains a comma separated list of name value pairs.

Remarks

The **CertificateSubject** property returns a string that contains information about the organization that the server certificate was issued to. The string value is a comma separated list of tagged name and value pairs. In the nomenclature of the X.500 standard, each of these pairs are called a relative distinguished name (RDN), and when concatenated together, forms the issuer's distinguished name (DN). For example:

```
C=US, O="RSA Data Security, Inc.", OU=Secure Server Certification Authority
```

To obtain a specific value, such as the name of the issuer or the issuer's country, the application must parse the string returned by this property. Some of the common tokens used in the distinguished name are:

Token	Description
C	The ISO standard two character country code.
S	The name of the state or province.
L	The name of the city or locality.
O	The name of the company or organization.
OU	The name of the department or organizational unit
CN	The common name; with X.509 certificates, this is the domain name of the site the certificate was issued for.

This property will return an empty string if a secure connection has not been established with the remote host.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.CipherStrength Property

Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.

[Visual Basic]
```
Public ReadOnly Property CipherStrength As Integer
```

[C#]
```
public int CipherStrength {get;}
```

Property Value

An integer value which specifies the encryption key length if a secure connection has been established; otherwise a value of 0 is returned.

Remarks

The **CipherStrength** property returns the number of bits in the key used to encrypt the secure data stream. Common values returned by this property are 128 and 256. A key length of 40 or 56 bits is considered insecure and subject to brute force attacks. 128-bit and 256-bit keys are considered secure. If this property returns a value of 0, this means that a secure connection has not been established with the remote host.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.CodePage Property

Gets and sets the code page used when reading and writing text.

[Visual Basic]
```
Public Property CodePage As Integer
```

[C#]
```
public int CodePage {get; set;}
```

Property Value

An integer value which specifies the current code page. A value of zero specifies the default code page for the current locale should be used. To preserve the original Unicode text, you can use code page 65001 which specifies UTF-8 character encoding.

Remarks

All data which is exchanged over a socket is sent and received as 8-bit bytes, typically referred to as "octets" in networking terminology. However, strings in .NET are Unicode where each character is represented by 16 bits. To send and receive data using strings, these Unicode strings are converted to a stream of bytes.

By default, strings are converted to an array of bytes using the code page for the current locale, mapping the 16-bit Unicode characters to bytes. Similarly, when reading data from the socket into a string buffer, the stream of bytes received from the remote host are converted to Unicode before they are returned to your application.

If you are exchanging text with another system and it appears to corrupted or characters are being replaced with question marks or other symbols, it is likely the system is sending text which is using a different character encoding. Most services use UTF-8 encoding to represent non-ASCII characters and selecting the UTF-8 code page will typically resolve the issue.

Strings are only guaranteed to be safe when sending and receiving text. Using a string data type is not recommended when reading or writing binary data to a socket. If possible, you should always use a byte array as the buffer parameter for the Read and Write methods whenever you are exchanging binary data.

For backwards compatibility, this class defaults to using the code page for the current locale. This property value directly corresponds to Windows code page identifiers, and will accept any valid code page supported by the .NET Framework. Setting this property to an invalid code page will generate an exception.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.ExternalAddress Property

Gets a value that specifies the external Internet address for the local system.

[Visual Basic]

```
Public ReadOnly Property ExternalAddress As String
```

[C#]

```
public string ExternalAddress {get;}
```

Property Value

A string which specifies an Internet address using dotted notation.

Remarks

The **ExternalAddress** property returns the IP address assigned to the router that connects the local host to the Internet. This is typically used by an application executing on a system in a local network that uses a router which performs Network Address Translation (NAT). In that network configuration, the **LocalAddress** property will only return the IP address for the local system on the LAN side of the network unless a connection has already been established to a remote host. The **ExternalAddress** property can be used to determine the IP address assigned to the router on the Internet side of the connection and can be particularly useful for servers running on a system behind a NAT router.

Using this property requires that you have an active connection to the Internet; checking the value of this property on a system that uses dial-up networking may cause the operating system to automatically connect to the Internet service provider. The class may be unable to determine the external IP address for the local host for a number of reasons, particularly if the system is behind a firewall or uses a proxy server that restricts access to external sites on the Internet. If the external address for the local host cannot be determined, the property will return an empty string.

If the class is able to obtain a valid external address for the local host, that address will be cached for sixty minutes. Because dial-up connections typically have different IP addresses assigned to them each time the system is connected to the Internet, it is recommended that this property only be used in conjunction with broadband connections using a NAT router.

It is important to note that checking this property value may cause the current thread to block until the external IP address can be resolved and should never be used in conjunction with non-blocking (asynchronous) socket connections. If you need to check this property value in an application which uses asynchronous sockets, it is recommended that you create a new thread and access the property from within that thread.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Handle Property

Gets a value that specifies the socket handle allocated for the current session.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer which represents a socket handle. If there is no active connection, a value of -1 is returned.

Remarks

The **Handle** property specifies the socket descriptor of the listening socket. To accept the connection, a new instance of the SocketWrench class should be created, passing this value to the **Accept** method in the new class instance.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.HashStrength Property

Gets a value which specifies the length of the message digest that was selected for a secure connection.

[Visual Basic]

```
Public ReadOnly Property HashStrength As Integer
```

[C#]

```
public int HashStrength {get;}
```

Property Value

An integer value which specifies the length of the message digest if a secure connection has been established; otherwise a value of 0 is returned.

Remarks

The **HashStrength** property returns the number of bits used in the message digest (hash) that was selected. Common values returned by this property are 128 and 160. If this property returns a value of 0, this means that a secure connection has not been established with the remote host.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.HostAddress Property

Gets and sets a value which specifies the Internet address used to establish a connection.

[Visual Basic]

```
Public Property HostAddress As String
```

[C#]

```
public string HostAddress {get; set;}
```

Property Value

A string which specifies an Internet address using dotted notation.

Remarks

The **HostAddress** property can be used to set the Internet address for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the address is assigned to a valid host name, the **HostName** property will be updated with that value.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.HostFile Property

Gets and sets a value that specifies the name of a host file used to resolve host names and addresses.

[Visual Basic]
```
Public Property HostFile As String
```

[C#]
```
public string HostFile {get; set;}
```

Property Value

A string which specifies a file name.

Remarks

The **HostFile** property is used to specify the name of an alternate file for resolving hostnames and IP addresses. The host file is used as a database that maps an IP address to one or more hostnames, and is used when setting the **HostName** or **HostAddress** properties and establishing a connection with a remote host. The file is a plain text file, with each line in the file specifying a record, and each field separated by spaces or tabs. The format of the file must be as follows:

```
ipaddress hostname [hostalias ...]
```

For example, one typical entry maps the name "localhost" to the local loopback IP address. This would be entered as:

```
127.0.0.1 localhost
```

The hash character (#) may be used to specify a comment in the file, and all characters after it are ignored up to the end of the line. Blank lines are ignored, as are any lines which do not follow the required format.

Setting this property loads the file into memory allocated for the current thread. If the contents of the file have changed after the function has been called, those changes will not be reflected when resolving hostnames or addresses. To reload the host file from disk, set the property again with the same file name. To remove the alternate host file from memory, specify an empty string as the file name.

If a host file has been specified, it is processed before the default host file when resolving a hostname into an IP address, or an IP address into a hostname. If the host name or address is not found, or no host file has been specified, a nameserver lookup is performed.

Because the alternate host file is cached for the current thread, setting this property will affect all instances of the class in the same thread. For example, if a project has created three instances of the class, setting the HostFile property will affect all three instances, not just the instance that set the property. To determine if an alternate host file has been cached, check the property value. If the property returns an empty string, no alternate host file has been cached.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.HostName Property

Gets and sets a value which specifies the host name used to establish a connection.

[Visual Basic]
```
Public Property HostName As String
```

[C#]
```
public string HostName {get; set;}
```

Property Value

A string which specifies a host name.

Remarks

The **HostName** property can be used to set the host name for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the name can be resolved to a valid Internet address, the **HostAddress** property will be updated with that value.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.InLine Property

Gets and sets a value that indicates if urgent data is received in-line with non-urgent data.

[Visual Basic]

```
Public Property InLine As Boolean
```

[C#]

```
public bool InLine {get; set;}
```

Property Value

Returns **true** if urgent data will be received in-line; otherwise returns **false**. The default value is **false**.

Remarks

The **InLine** property controls how urgent (out-of-band) data is handled when reading data from the socket. If set to a value of **true**, urgent data is placed in the data stream along with non-urgent data. To determine if the data that is being read is urgent, the **AtMark** property can be read.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.IsBlocked Property

Gets a value which indicates if the current thread is performing a blocking socket operation.

[Visual Basic]

```
Public ReadOnly Property IsBlocked As Boolean
```

[C#]

```
public bool IsBlocked {get;}
```

Property Value

Returns **true** if the current thread is blocking, otherwise returns **false**.

Remarks

The **IsBlocked** property returns **true** if the current thread is blocked performing an operation. Because the Windows Sockets API only permits one blocking operation per thread of execution, this property should be checked before starting any blocking operation in response to an event.

If the **IsBlocked** property returns **false**, this means there are no blocking operations on the current thread at that time. However, this does not guarantee that the next socket operation will not fail. An application should always check the return value from a socket operation and check the value of the **LastError** property if an error occurs.

Note that this property will return **true** if there is any blocking operation being performed by the current thread, regardless of whether this specific instance of the class is responsible for the blocking operation or not.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.IsClosed Property

Gets a value which indicates if the connection to the remote host has been closed.

[Visual Basic]

```
Public ReadOnly Property IsClosed As Boolean
```

[C#]

```
public bool IsClosed {get;}
```

Property Value

Returns **true** if the connection has been closed; otherwise returns **false**.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.IsConnected Property

Gets a value which indicates if a connection to the remote host has been established.

[Visual Basic]

```
Public ReadOnly Property IsConnected As Boolean
```

[C#]

```
public bool IsConnected {get;}
```

Property Value

Returns **true** if the connection has been established; otherwise returns **false**.

Remarks

The **IsConnected** property can only be used to indicate if there is still a logical connection to the remote host. It cannot be used to detect abnormal conditions such as the remote host aborting the connection, the physical network connection being lost or other critical errors.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.IsInitialized Property

Gets a value which indicates if the current instance of the class has been initialized successfully.

[Visual Basic]
```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]
```
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the class instance has been initialized; otherwise returns **false**.

Remarks

The **IsInitialized** property is used to determine if the current instance of the class has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the class may not be able to initialize itself.

The most common reasons that a class instance may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required networking libraries or not being able to access the system registry.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.IsListening Property

Gets a value which indicates if the socket is listening for client connections.

[Visual Basic]

```
Public ReadOnly Property IsListening As Boolean
```

[C#]

```
public bool IsListening {get;}
```

Property Value

Returns **true** if the socket is listening for client connections; otherwise returns **false**.

Remarks

The **IsListening** property will return **true** if the socket was created using the **Listen** method and it is currently accepting incoming client connections. In all other situations, this property will return **false**.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.IsReadable Property

Gets a value which indicates if there is data available to be read from the socket.

[Visual Basic]

```
Public ReadOnly Property IsReadable As Boolean
```

[C#]

```
public bool IsReadable {get;}
```

Property Value

Returns **true** if there is data available to be read; otherwise returns **false**.

Remarks

The **IsReadable** property returns **true** if data can be read from the socket without blocking. For non-blocking sockets, this property can be checked before the application attempts to read the socket. Note that even if this property does return **true** indicating that there is data available to be read, applications should always check the return value from the **Read** method.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.IsWritable Property

Gets a value which indicates if data can be written to the socket without blocking.

[Visual Basic]

```
Public ReadOnly Property IsWritable As Boolean
```

[C#]

```
public bool IsWritable {get;}
```

Property Value

Returns **true** if data can be written to the socket; otherwise returns **false**.

Remarks

The **IsWritable** property returns **true** if data can be written to the socket without blocking. For non-blocking sockets, this property can be checked before the application attempts to write data to the socket. Note that even if this property does return **true** indicating that data can be written to the socket, applications should always check the return value from the **Write** method.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.KeepAlive Property

Gets and sets a value which indicates if keep-alive packets are sent on a connected socket.

[Visual Basic]
```
Public Property KeepAlive As Boolean
```

[C#]
```
public bool KeepAlive {get; set;}
```

Property Value

Returns **true** if keep-alive packets are sent when the connection is idle, otherwise returns **false**. The default value is **false**.

Remarks

Setting the **KeepAlive** property to a value of **true** specifies that special packets are to be sent to the remote system when no data is being exchanged to ensure the connection remains active. This property can only be set for sockets that were created with the **Protocol** property set to a value of **SocketProtocol.protocolStream**.

If this property is set to **true**, keep-alive packets will start being generated five seconds after the socket has become idle with no data being sent or received. Enabling this option can be used by applications to detect when a physical network connection has been lost. However, it is recommended that most applications query the remote host directly to determine if the connection is still active. This is typically accomplished by sending specific commands to the server to query its status, or checking the elapsed time since the last response from the server.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public SocketWrench.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Linger Property

Gets and sets a value which specifies the number of seconds to wait for the socket to disconnect from the remote host.

[Visual Basic]

```
Public Property Linger As Integer
```

[C#]

```
public int Linger {get; set;}
```

Property Value

An integer value which specifies a number of seconds. The default value is 0.

Remarks

Setting the **Linger** property to a value greater than zero indicates that the **Disconnect** method should wait up to the specified number of seconds for any data on the socket to be written before it is closed. A value of zero indicates that the socket should be closed immediately (but gracefully, without data loss).

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.LocalAddress Property

Gets and sets the local Internet address that the socket will be bound to.

[Visual Basic]

```
Public Property LocalAddress As String
```

[C#]

```
public string LocalAddress {get; set;}
```

Property Value

A string which specifies an Internet address in dotted notation.

Remarks

The **LocalAddress** property is used to specify the local Internet address that the socket will be bound to when a connection is established with a remote host. By default this property is not assigned a value, which specifies that the socket should be bound to any appropriate network interface on the local system.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.LocalName Property

Gets a value which specifies the host name for the local system.

[Visual Basic]

```
Public ReadOnly Property LocalName As String
```

[C#]

```
public string LocalName {get;}
```

Property Value

A string which specifies the local host name.

Remarks

The **LocalName** property returns the fully-qualified host name assigned to the local system. If the system has not been configured with an Internet domain name, then this property will return the NetBIOS name assigned to the local system.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.LocalPort Property

Gets and sets a value which specifies the local port number the socket will be bound to.

[Visual Basic]

```
Public Property LocalPort As Integer
```

[C#]

```
public int LocalPort {get; set;}
```

Property Value

An integer value which specifies a port number. The default value is 0.

Remarks

The **LocalPort** property is used to specify the local port number that the socket will be bound to when a connection is established with a remote host. By default this property value is 0, which specifies that the socket should be bound to any appropriate port number that is available on the local system. After a connection has been established, this property will return the actual port number that was allocated for the socket.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.NoDelay Property

Gets and sets a value which specifies if the Nagle algorithm should be enabled or disabled.

[Visual Basic]

```
Public Property NoDelay As Boolean
```

[C#]

```
public bool NoDelay {get; set;}
```

Property Value

Returns **true** if the Nagle algorithm has been disabled; otherwise it returns **false**. The default value is **false**.

Remarks

The **NoDelay** property is used to enable or disable the Nagle algorithm, which buffers unacknowledged data and insures that a full-size packet can be sent to the remote host. By default this property value is set to **false**, which enables the Nagle algorithm (in other words, the data being written may not actually be sent until it is optimal to do so). Setting this property to **true** disables the Nagle algorithm, maintaining the time delays between the data packets being sent.

This property should be set to **true** only if it is absolutely required and the implications of doing so are understood. Disabling the Nagle algorithm can have a significant negative impact on the performance of your application.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Options Property

Gets and sets a value which specifies one or more socket options.

[Visual Basic]

```
Public Property Options As SocketOptions
```

[C#]

```
public SocketWrench.SocketOptions Options {get; set;}
```

Property Value

Returns one or more SocketOptions enumeration flags which specify the options for the socket. The default value for this property is **socketOptionNone**.

Remarks

The **Options** property specifies one or more default socket options which are used when creating a socket using either the **Accept** or **Connect** methods.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.PeerAddress Property

Gets a value that specifies the Internet address of the remote host.

[Visual Basic]

```
Public ReadOnly Property PeerAddress As String
```

[C#]

```
public string PeerAddress {get;}
```

Property Value

A string which specifies an Internet address in dotted notation.

Remarks

The **PeerAddress** property returns the Internet address of the remote system that the local host is connected to. If a datagram socket is being used, this property will return the address of the system which sent the last datagram that was read. If no connection has been established, this property will return an empty string.

If this property is accessed inside an **OnAccept** event handler, it will return the address of the client that is requesting the connection. The application may use this information to determine if it wishes to accept or reject the client connection. If the address is not available to the client at that time, this property will return the address 0.0.0.0.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.PeerName Property

Gets a value that specifies the name of the remote host.

[Visual Basic]

```
Public ReadOnly Property PeerName As String
```

[C#]

```
public string PeerName {get;}
```

Property Value

A string which specifies the peer host name.

Remarks

The **PeerName** property returns the name of the remote system that the local host is connected to. If a datagram socket is being used, this property will return the name of the system which sent the last datagram that was read.

Accessing this property may cause the thread to block until the peer address can be resolved to a host name.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.PeerPort Property

Gets a value that specifies the port number used by the remote host.

[Visual Basic]

```
Public ReadOnly Property PeerPort As Integer
```

[C#]

```
public int PeerPort {get;}
```

Property Value

An integer value which specifies the peer port number.

Remarks

The **PeerName** property returns the port number of the remote system that the local host is connected to. If a datagram socket is being used, this property will return the port number of the remote host which sent the last datagram that was read.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.PhysicalAddress Property

Gets a value which specifies the MAC address for the local system's network adapter.

[Visual Basic]

```
Public ReadOnly Property PhysicalAddress As String
```

[C#]

```
public string PhysicalAddress {get;}
```

Property Value

A string which specifies the network adapter MAC address.

Remarks

The **PhysicalAddress** property returns the Media Access Control (MAC) address for an Ethernet or Token Ring network adapter installed and configured on the local system. Since it is guaranteed that every adapter is assigned a unique address throughout the world, this value can be safely used for identification purposes. It is possible that this property will return an empty string, which indicates that it could not find a network adapter.

If more than one physical network adapter is installed on the system, this property will return the MAC address of the first adapter that it finds.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Protocol Property

Gets and sets a value which specifies the socket protocol.

[Visual Basic]
```
Public Property Protocol As SocketProtocol
```

[C#]
```
public SocketWrench.SocketProtocol Protocol {get; set;}
```

Property Value

Returns a SocketProtocol enumeration value which specifies the socket protocol. The default value is **socketStream**.

Remarks

The **Protocol** property specifies the type of socket that will be created. This property may only be set before the **Connect** method is called; attempting to change this property value after a connection has been established will generate an error.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.RemotePort Property

Gets and sets a value which specifies the remote port number.

[Visual Basic]

```
Public Property RemotePort As Integer
```

[C#]

```
public int RemotePort {get; set;}
```

Property Value

An integer value which specifies a port number.

Remarks

The **RemotePort** property is used to set the port number that will be used to establish a connection with a remote host. If the port number specifies a well-known port, the **RemoteService** property will be updated with that name.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.ReservedPort Property

Gets and sets a value which indicates if a reserved port number was used.

[Visual Basic]
```
Public Property ReservedPort As Boolean
```

[C#]
```
public bool ReservedPort {get; set;}
```

Property Value

Returns **true** if a reserved port number was used; otherwise returns **false**. The default value is **false**.

Remarks

The **ReservedPort** property determines if a reserved local port number is use when the socket is created (reserved port numbers are in the range of 513 through 1023, inclusive). Some application protocols require that the client bind to a local port number in this range. By setting the **LocalPort** property to 0 and the **ReservedPort** property to **true**, a reserved port number will be used when the socket is created. The default value for this property is **false**, which specifies that a standard port number with a value of 1024 or higher will be bound to the socket unless the **LocalPort** property is explicitly set to a non-zero value. Reserved ports should only be used by those applications that need them to implement a specific protocol.

It is possible that the error **errorAddressInUse** will be returned when attempting to connect using a reserved port number. The value of the **LocalPort** property will specify the reserved port number that could not be used.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.ReuseAddress Property

Gets and sets a value which indicates if a socket address can be reused.

[Visual Basic]

```
Public Property ReuseAddress As Boolean
```

[C#]

```
public bool ReuseAddress {get; set;}
```

Property Value

Returns **true** if an address can be reused; otherwise returns **false**. The default value is **true**.

Remarks

The **ReuseAddress** property determines if a socket can be bound to an address and port number that were recently used. If this property is **true**, then addresses can be reused as needed. If the property is **false**, then addresses cannot be reused and an error will be generated if the address was was recently used by another socket.

This property is typically used by server applications. By setting the property to **true**, a server can be stopped and immediately restarted using the same port number; otherwise, the server must wait approximately two minutes before the port can be reused.

If you wish to determine if a local port number is already in use by another application, set this property to **false** and attempt to create a socket using that port number. If another application is already using that port number, an error will be generated indicating that the address is in use and the socket could not be created.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Route Property

Gets and sets a value which indicates if packets should be routed.

[Visual Basic]

```
Public Property Route As Boolean
```

[C#]

```
public bool Route {get; set;}
```

Property Value

Returns **true** if packets should be routed; otherwise returns **false**. The default value is **true**.

Remarks

The **Route** property determines if routing tables should be used when sending data. If the property is set to **false**, then packets will be sent directly to the network interface; if there is a router between the local and remote hosts, the data will be lost. It is not recommended that you change this property value unless it is required by your application and you fully understand the implications of doing so.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Secure Property

Gets and sets a value which specifies if a secure connection is established.

[Visual Basic]
```
Public Property Secure As Boolean
```

[C#]
```
public bool Secure {get; set;}
```

Property Value

Returns **true** if a secure connection is established; otherwise returns **false**. The default value is **false**.

Remarks

The **Secure** property determines if a secure connection is established with the remote host. The default value for this property is **false**, which specifies that a standard connection to the server is used. To establish a secure connection, the application should set this property value to **true** prior to calling the **Accept** or **Connect** methods. Once the connection has been established, the client may exchange data with the server as with standard connections.

It is possible for an application to establish a non-secure connection, and then switch to a secure connection at some later point during the session. Initially set the **Secure** property to **false**, then connect to the server normally. Once the connection has been established, setting the **Secure** property to true will cause the application to negotiate a secure connection with the remote host. If the socket was created using the **Accept** method, the class will block and wait for the client to begin the negotiation. If the socket was created using the **Connect** method, it will immediately begin the negotiation with the server. Note that if a non-blocking (asynchronous) socket is being used, the application must wait to set the **Secure** property to **true** after the **OnConnect** event has fired.

Setting the **Secure** property to **false** during a connection will cause the class to send a shutdown message to the remote host. This may cause the remote host to terminate the connection, however it will not close the socket. It is recommended that applications do not set the **Secure** property to **false** after a secure connection has been established, and instead use the **Disconnect** method to close the connection.

It is strongly recommended that any application that sets this property **true** use error handling to trap an errors that may occur. If the class is unable to initialize the security libraries, or otherwise cannot create a secure session for the client, an error will be generated when this property value is set.

Example

```
Socket.ThrowError = True

Try
    Socket.Secure = True
    Socket.Connect(strHostName, nHostPort, defTimeout)

    Socket.WriteLine("GET " + strFileName + " HTTP/1.0")
    Socket.WriteLine("Host: " + strHostName)
    Socket.WriteLine("Accept: text/*")
    Socket.WriteLine()

    Do
        Socket.ReadLine(strBuffer)
    Loop Until strBuffer.Length = 0

    Socket.ReadStream(strBuffer, True)
```

```
Catch ex As SocketTools.SocketWrenchException
    MsgBox(ex.Message)
End Try

Socket.Disconnect()
```

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.SecureCipher Property

Gets a value that specifies the encryption algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureCipher As SecureCipherAlgorithm
```

[C#]

```
public SocketWrench.SecureCipherAlgorithm SecureCipher {get;}
```

Property Value

A SecureCipherAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureCipher** property returns a value which identifies the algorithm used to encrypt the data stream. If a secure connection has not been established, this property will return a value of **cipherNone**.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.SecureHash Property

Gets a value that specifies the message digest algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureHash As SecureHashAlgorithm
```

[C#]

```
public SocketWrench.SecureHashAlgorithm SecureHash {get;}
```

Property Value

A SecureHashAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureHash** property returns a value which identifies the message digest (hash) algorithm that was selected when a secure connection was established. If a secure connection has not been established, this property will return a value of **hashNone**.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.SecureKeyExchange Property

Gets a value that specifies the key exchange algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureKeyExchange As SecureKeyAlgorithm
```

[C#]

```
public SocketWrench.SecureKeyAlgorithm SecureKeyExchange {get;}
```

Property Value

A SecureKeyAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureKeyExchange** property returns a value which identifies the key exchange algorithm that was selected when a secure connection was established. If a secure connection has not been established, this property will return a value of **keyExchangeNone**.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.SecureProtocol Property

Gets and sets a value which specifies the protocol used for a secure connection.

[Visual Basic]
```
Public Property SecureProtocol As SecurityProtocols
```

[C#]
```
public SocketWrench.SecurityProtocols SecureProtocol {get; set;}
```

Property Value

A SecurityProtocols enumeration value which identifies the protocol to be used when establishing a secure connection.

Remarks

The **SecureProtocol** property can be used to specify the security protocol to be used when establishing a secure connection with a server or accepting a secure connection from a client. By default, the class will attempt to use either SSL v3 or TLS v1 to establish the connection, with the appropriate protocol automatically selected based on the capabilities of the remote host. It is recommended that you only change this property value if you fully understand the implications of doing so. Assigning a value to this property will override the default protocol and force the class to attempt to use only the protocol specified.

Multiple security protocols may be specified by combining them using a bitwise **or** operator. After a connection has been established, this property will identify the protocol that was selected. Attempting to set this property after a connection has been established will result in an exception being thrown. This property should only be set after setting the **Secure** property to **true** and before calling the **Accept** or **Connect** methods.

In some cases, a server may only accept a secure connection if the TLS v1 protocol is specified. If the security protocol is not compatible with the server, then the connection will fail with an error indicating that the class is unable to establish a security context for the session. In this case, try assigning the property to **protocolTLS1** and attempt the connection again.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Status Property

Gets a value which specifies the current status of the socket.

[Visual Basic]

```
Public ReadOnly Property Status As SocketStatus
```

[C#]

```
public SocketWrench.SocketStatus Status {get;}
```

Property Value

A SocketStatus enumeration value which specifies the current socket status.

Remarks

The **Status** property returns the current status of the socket. This property should be checked on blocking sockets to determine if the socket is in use before taking some action.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.ThreadModel Property

Gets and sets a value which specifies the threading model for the class instance.

[Visual Basic]

```
Public Property ThreadModel As ThreadingModel
```

[C#]

```
public SocketWrench.ThreadingModel ThreadModel {get; set;}
```

Property Value

Returns one or more ThreadingModel enumeration value which specifies the threading model for the client. The default value for this property is **modelSingleThread**.

Remarks

The **ThreadModel** property specifies the threading model that is used by the class instance when a connection is established. The default value for this property is **modelSingleThread**, which specifies that only the thread that established the connection should be permitted to invoke methods. It is important to note that this threading model does not limit the application to a single thread of execution. When a session is established using the **Connect** method, that session is attached to the thread that created it. From that point on, until the session is terminated, only the owner may invoke methods in that instance of the class. The ownership of the class instance may be transferred from one thread to another using the **AttachThread** method.

Setting this property to **modelFreeThread** disables certain internal safety checks that are performed by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads. The application must ensure that no two threads will attempt to invoke a blocking method at the same time. In other words, if one thread invokes a method, the application must ensure that another thread will not attempt to invoke any other method at the same time using the same instance of the class.

Changing the value of this property will not affect an active client session. The threading model must be specified prior to invoking the **Connect** method.

See Also

SocketWrench Class | SocketTools Namespace | AttachThread Method | ThreadingModel Enumeration | ThreadModel Attribute

SocketWrench.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]

```
Public Property ThrowError As Boolean
```

[C#]

```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Timeout Property

Gets and sets a value which specifies a timeout period in seconds.

[Visual Basic]
```
Public Property Timeout As Integer
```

[C#]
```
public int Timeout {get; set;}
```

Property Value

An integer value which specifies a timeout period in seconds.

Remarks

Setting the **Timeout** property specifies the number of seconds until a blocking socket operation fails and returns an error.

The timeout period is only used when the socket is in blocking mode. Although this property can be changed when the socket is in non-blocking mode, the value will be ignored until the socket is returned to blocking mode.

For most applications it is recommended the timeout period be set between 10 and 20 seconds.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Trace Property

Gets and sets a value which indicates if network function logging is enabled.

[Visual Basic]
```
Public Property Trace As Boolean
```

[C#]
```
public bool Trace {get; set;}
```

Property Value

Returns **true** if network function tracing is enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Trace** property is used to enable (or disable) the tracing of network function calls. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the SocketTools networking classes and is enabled or disabled for an entire process. This means that once trace logging is enabled for a given component, all of the function calls made by the process using any of the SocketTools classes will be logged. For example, if you have an application using both the File Transfer Protocol and Post Office Protocol classes, and you set the **Trace** property to **true**, function calls made by both classes will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all classes used by the process.

If trace logging is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the logfile is fairly large. Since logfiles can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

When redistributing your application, make sure that you include the **SocketTools10.TraceLog.dll** module with your installation. If this library is not present, then no trace output will be generated and the value of the **Trace** property will be ignored. Only those function calls made by the SocketTools networking classes will be logged. Calls made directly to the Windows Sockets API, or calls made by other classes, will not be logged.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.TraceFile Property

Gets and sets a value which specifies the name of the network function tracing logfile.

[Visual Basic]

```
Public Property TraceFile As String
```

[C#]

```
public string TraceFile {get; set;}
```

Property Value

A string which specifies the name of the file.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named **SocketTools.log** is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFile** property should be set to the same value for each instance. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
 MyApp INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0 MyApp WRN:
connect(46, 192.0.0.1:1234, 16) returned -1 [10035] MyApp ERR: accept(46,
NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the class is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Note that if the specified file cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.TraceFlags Property

Gets and sets a value which specifies the network function tracing flags.

[Visual Basic]

```
Public Property TraceFlags As TraceOptions
```

[C#]

```
public SocketWrench.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration which specifies the amount of detail written to the trace logfile.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Because network function tracing is enabled per-process, the trace flags are shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFlags** property should be set to the same value for each instance. Changing the trace flags for any one instance of the class will affect the logging performed for all SocketTools classes used by the application.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being written and the error **errorOperationWouldBlock** occurs, a warning is generated because the application simply needs to attempt to write the data at a later time.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Urgent Property

Gets and sets a value which specifies if urgent data will be read or written.

[Visual Basic]

```
Public Property Urgent As Boolean
```

[C#]

```
public bool Urgent {get; set;}
```

Property Value

Returns **true** if urgent data will be read or written; otherwise returns **false**. The default value is **false**.

Remarks

The **Urgent** property affects how the **Read** and **Write** methods receive and transmit data to the remote host. If set to a value of **true**, urgent (out-of-band) data will be read or written. The property value will automatically be reset to a value of **false** after the data has been read or written.

It is important to note that all systems may support more than one byte of urgent data if the data is not being received in-line. Refer to the **InLine** property for additional information. This property should only be set to **true** if required by the application and the implications of doing so are understood.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Version Property

Gets a value which returns the current version of the SocketWrench class library.

[Visual Basic]

```
Public ReadOnly Property Version As String
```

[C#]

```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the SocketWrench class library. This value can be used by an application for validation and debugging purposes.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench Methods

The methods of the **SocketWrench** class are listed below. For a complete list of **SocketWrench** class members, see the SocketWrench Members topic.

Public Instance Methods

Abort	Abort the connection with a remote host.
Accept	Overloaded. Accepts a client connection on a listening socket, specifying a timeout period and one or more socket options.
AttachThread	Attach an instance of the class to the current thread
Bind	Overloaded. Bind the socket to the specified local address and port number.
Cancel	Cancel the current blocking socket operation.
Connect	Overloaded. Establish a connection with a remote host.
Disconnect	Terminate the connection with a remote host.
Dispose	Overloaded. Releases all resources used by SocketWrench.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the SocketWrench class.
Listen	Overloaded. Listen for incoming client connections, specifying the local network address, port number and connection backlog.
Peek	Overloaded. Read data from the socket and store it in a byte array, but do not remove the data from the socket buffers.
Read	Overloaded. Read data from the socket and store it in a byte array.
ReadFrom	Overloaded. Read data from the socket and store it in a byte array.
ReadLine	Overloaded. Read up to a line of data from the socket and return it in a string buffer.
ReadStream	Overloaded. Read a data stream from the socket and store it in the specified byte array.

Reject	Rejects a connection request from a remote host.
Reset	Reset the internal state of the object, resetting all properties to their default values.
Resolve	Resolves a host name to a host IP address.
Shutdown	Overloaded. Disable sending or receiving data on the socket.
StoreStream	Overloaded. Reads a data stream from the socket and stores it in the specified file.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
Write	Overloaded. Write one or more bytes of data to the socket.
WriteLine	Overloaded. Send a line of text to the remote host, terminated by a carriage-return and linefeed.
WriteStream	Overloaded. Write a stream of bytes to the socket.
WriteTo	Overloaded. Write one or more bytes of data to the socket.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the SocketWrench class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Abort Method

Abort the connection with a remote host.

[Visual Basic]

```
Public Sub Abort()
```

[C#]

```
public void Abort();
```

Remarks

The **Abort** method immediately closes the socket, without waiting for any remaining data to be written out. This method should only be used when the connection must be closed immediately. If this method is used, the remote host will see the connection as being terminated abnormally.

It is recommended that applications using the **Disconnect** method unless it is absolutely necessary to terminate the connection and immediately release the socket handle.

See Also

SocketWrench Class | SocketTools Namespace | Disconnect Method

SocketWrench.Accept Method

Accepts a client connection on a listening socket.

Overload List

Accepts a client connection on a listening socket.

public bool Accept(int);

Accepts a client connection on a listening socket, specifying one or more socket options.

public bool Accept(int,SocketOptions);

Accepts a client connection on a listening socket, specifying a timeout period and one or more socket options.

public bool Accept(int,int,SocketOptions);

See Also

SocketWrench Class | SocketTools Namespace | Listen Method

SocketWrench.Accept Method (Int32)

Accepts a client connection on a listening socket.

[Visual Basic]

```
Overloads Public Function Accept( _
   ByVal handle As Integer _
) As Boolean
```

[C#]

```
public bool Accept(
   int handle
);
```

Parameters

handle

The socket identifier of a listening socket. If the object that invokes this method is not the listening socket, then the listening socket may continue to listen for incoming connections. If the object of a listening socket invokes this method with its own handle, then it ceases to listen, and no other host can establish a connection with the application.

Return Value

A boolean value which specifies if the client connection has been accepted. If the method returns **true**, the connection has been accepted and the application may send and receive data with the remote host. If this method returns **false**, the connection could not be accepted and the application should check the value of the **LastError** property to determine the cause of the failure.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Accept Overload List | Listen Method

SocketWrench.Accept Method (Int32, SocketOptions)

Accepts a client connection on a listening socket, specifying one or more socket options.

[Visual Basic]

```
Overloads Public Function Accept( _
   ByVal handle As Integer, _
   ByVal options As SocketOptions _
) As Boolean
```

[C#]

```
public bool Accept(
   int handle,
   SocketOptions options
);
```

Parameters

handle

The socket identifier of a listening socket. If the object that invokes this method is not the listening socket, then the listening socket may continue to listen for incoming connections. If the object of a listening socket invokes this method with its own handle, then it ceases to listen, and no other host can establish a connection with the application.

options

One or more of the SocketOptions enumeration flags.

Return Value

A boolean value which specifies if the client connection has been accepted. If the method returns **true**, the connection has been accepted and the application may send and receive data with the remote host. If this method returns **false**, the connection could not be accepted and the application should check the value of the **LastError** property to determine the cause of the failure.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Accept Overload List | Listen Method

SocketWrench.Accept Method (Int32, Int32, SocketOptions)

Accepts a client connection on a listening socket, specifying a timeout period and one or more socket options.

[Visual Basic]

```
Overloads Public Function Accept( _
   ByVal handle As Integer, _
   ByVal timeout As Integer, _
   ByVal options As SocketOptions _
) As Boolean
```

[C#]

```
public bool Accept(
   int handle,
   int timeout,
   SocketOptions options
);
```

Parameters

handle

The socket identifier of a listening socket. If the object that invokes this method is not the listening socket, then the listening socket may continue to listen for incoming connections. If the object of a listening socket invokes this method with its own handle, then it ceases to listen, and no other host can establish a connection with the application.

timeout

Specifies the number of seconds that the method will wait for a client connection to be established on the listening socket. This value only has meaning for a blocking socket.

options

One or more of the SocketOptions enumeration flags.

Return Value

A boolean value which specifies if the client connection has been accepted. If the method returns **true**, the connection has been accepted and the application may send and receive data with the remote host. If this method returns **false**, the connection could not be accepted and the application should check the value of the LastError property to determine the cause of the failure.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Accept Overload List | Listen Method

SocketWrench.AttachThread Method

Attach an instance of the class to the current thread

[Visual Basic]
```
Public Function AttachThread() As Boolean
```

[C#]
```
public bool AttachThread();
```

Return Value

A boolean value which specifies if the socket could be attached to the current thread. If this method returns **false**, the socket could not be attached to the thread and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

When an instance of the class is created it is associated with the current thread that created it. Normally, if another thread attempts to perform an operation using that instance, an error is returned. This is used to ensure that other threads cannot interfere with an operation being performed by the owner thread. In some cases, it may be desirable for one thread in an application to create an instance of the class, establish a connection and then pass that instance to another worker thread. The **AttachThread** method can be used to change the ownership of the class instance to the new worker thread.

This method should be called by the new thread immediately after it has been created, and if the new thread does not release the handle itself, the ownership of the handle should be restored by the original thread. Under no circumstances should **AttachThread** be used to forcibly destroy an instance of a class allocated by another thread while a blocking operation is in progress. To cancel a blocking operation, use the **Cancel** method and then delete the class instance after the blocking function exits and control is returned to the current thread.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Bind Method

Bind the socket to the specified local address and port number.

Overload List

Bind the socket to the specified local address and port number.

public bool Bind(string,int);

Bind the socket to the specified local address and port number.

public bool Bind(string,int,SocketOptions);

Bind the socket to the specified local address and port number.

public bool Bind(string,int,SocketProtocol);

Bind the socket to the specified local address and port number.

public bool Bind(string,int,SocketProtocol,SocketOptions);

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Bind Method (String, Int32)

Bind the socket to the specified local address and port number.

[Visual Basic]

```
Overloads Public Function Bind( _
   ByVal localAddress As String, _
   ByVal localPort As Integer _
) As Boolean
```

[C#]

```
public bool Bind(
   string localAddress,
   int localPort
);
```

Parameters

localAddress

A string which specifies the local Internet address that the socket should be bound to. To bind to any valid network interface on the local system, specify the address 0.0.0.0. Applications should only specify a particular address if it is absolutely necessary. In most cases a local address is not required when establishing a client connection.

localPort

An integer value which specifies a local port number that the socket should be bound to. To bind to any available port number, specify a port number of 0. Applications should only specify a particular port number if it is absolutely necessary. The maximum valid port number is 65535.

Return Value

A boolean value which specifies if the socket could be bound to the specified address. If this method returns **false**, the socket could not be bound to the address and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Bind** method is used to specify the local address and port number that a socket will be bound to when it is created. When this method is called with **socketDatagram** as the specified protocol, it will immediately create the datagram socket and bind it to the given address.

When this method is called with **socketStream** as the specified protocol, creation of the socket is deferred until the **Connect** method is called. For stream sockets, this method will set the local address, port number and default options used when the socket is actually created.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Bind Overload List

SocketWrench.Bind Method (String, Int32, SocketOptions)

Bind the socket to the specified local address and port number.

[Visual Basic]

```
Overloads Public Function Bind( _
   ByVal localAddress As String, _
   ByVal localPort As Integer, _
   ByVal options As SocketOptions _
) As Boolean
```

[C#]

```
public bool Bind(
   string localAddress,
   int localPort,
   SocketOptions options
);
```

Parameters

localAddress

A string which specifies the local Internet address that the socket should be bound to. To bind to any valid network interface on the local system, specify the address 0.0.0.0. Applications should only specify a particular address if it is absolutely necessary. In most cases a local address is not required when establishing a client connection.

localPort

An integer value which specifies a local port number that the socket should be bound to. To bind to any available port number, specify a port number of 0. Applications should only specify a particular port number if it is absolutely necessary. The maximum valid port number is 65535.

options

One or more of the SocketOptions enumeration flags.

Return Value

A boolean value which specifies if the socket could be bound to the specified address. If this method returns **false**, the socket could not be bound to the address and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Bind** method is used to specify the local address and port number that a socket will be bound to when it is created. When this method is called with **socketDatagram** as the specified protocol, it will immediately create the datagram socket and bind it to the given address.

When this method is called with **socketStream** as the specified protocol, creation of the socket is deferred until the **Connect** method is called. For stream sockets, this method will set the local address, port number and default options used when the socket is actually created.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Bind Overload List

SocketWrench.Bind Method (String, Int32, SocketProtocol)

Bind the socket to the specified local address and port number.

[Visual Basic]

```
Overloads Public Function Bind( _
   ByVal localAddress As String, _
   ByVal localPort As Integer, _
   ByVal protocol As SocketProtocol _
) As Boolean
```

[C#]

```
public bool Bind(
   string localAddress,
   int localPort,
   SocketProtocol protocol
);
```

Parameters

localAddress

A string which specifies the local Internet address that the socket should be bound to. To bind to any valid network interface on the local system, specify the address 0.0.0.0. Applications should only specify a particular address if it is absolutely necessary. In most cases a local address is not required when establishing a client connection.

localPort

An integer value which specifies a local port number that the socket should be bound to. To bind to any available port number, specify a port number of 0. Applications should only specify a particular port number if it is absolutely necessary. The maximum valid port number is 65535.

protocol

One of the SocketProtocol enumeration values which specify the type of socket to be created.

Return Value

A boolean value which specifies if the socket could be bound to the specified address. If this method returns **false**, the socket could not be bound to the address and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Bind** method is used to specify the local address and port number that a socket will be bound to when it is created. When this method is called with **socketDatagram** as the specified protocol, it will immediately create the datagram socket and bind it to the given address.

When this method is called with **socketStream** as the specified protocol, creation of the socket is deferred until the **Connect** method is called. For stream sockets, this method will set the local address, port number and default options used when the socket is actually created.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Bind Overload List

SocketWrench.Bind Method (String, Int32, SocketProtocol, SocketOptions)

Bind the socket to the specified local address and port number.

[Visual Basic]

```
Overloads Public Function Bind( _
   ByVal localAddress As String, _
   ByVal localPort As Integer, _
   ByVal protocol As SocketProtocol, _
   ByVal options As SocketOptions _
) As Boolean
```

[C#]

```
public bool Bind(
   string localAddress,
   int localPort,
   SocketProtocol protocol,
   SocketOptions options
);
```

Parameters

localAddress

A string which specifies the local Internet address that the socket should be bound to. To bind to any valid network interface on the local system, specify the address 0.0.0.0. Applications should only specify a particular address if it is absolutely necessary. In most cases a local address is not required when establishing a client connection.

localPort

An integer value which specifies a local port number that the socket should be bound to. To bind to any available port number, specify a port number of 0. Applications should only specify a particular port number if it is absolutely necessary. The maximum valid port number is 65535.

protocol

One of the SocketProtocol enumeration values which specify the type of socket to be created.

options

One or more of the SocketOptions enumeration flags.

Return Value

A boolean value which specifies if the socket could be bound to the specified address. If this method returns **false**, the socket could not be bound to the address and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

The **Bind** method is used to specify the local address and port number that a socket will be bound to when it is created. When this method is called with **socketDatagram** as the specified protocol, it will immediately create the datagram socket and bind it to the given address.

When this method is called with **socketStream** as the specified protocol, creation of the socket is deferred until the **Connect** method is called. For stream sockets, this method will set the local address, port number and default options used when the socket is actually created.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Bind Overload List

SocketWrench.Cancel Method

Cancel the current blocking socket operation.

[Visual Basic]

```
Public Sub Cancel()
```

[C#]

```
public void Cancel();
```

Remarks

When the **Cancel** method is called, the blocking socket operation will not immediately fail. An internal flag is set which causes the blocking operation to exit with an error. This means that the application cannot cancel an operation and immediately perform some other blocking function. Instead it must allow the calling stack to unwind, returning back to the blocking operation before making any further function calls.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Connect Method

Establish a connection with a remote host.

Overload List

Establish a connection with a remote host.

public bool Connect();

Establish a connection with a remote host.

public bool Connect(string,int);

Establish a connection with a remote host.

public bool Connect(string,int,SocketProtocol,int);

Establish a connection with a remote host.

public bool Connect(string,int,SocketProtocol,int,SocketOptions);

Establish a connection with a remote host.

public bool Connect(string,int,SocketProtocol,int,SocketOptions,string,int);

Establish a connection with a remote host.

public bool Connect(string,int,int);

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Connect Method ()

Establish a connection with a remote host.

[Visual Basic]
```
Overloads Public Function Connect() As Boolean
```

[C#]
```
public bool Connect();
```

Return Value

A boolean value which specifies if the connection has been established. If the socket is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the socket is in non-blocking mode, a return value of **true** indicates that the socket has been successfully created and the connection is in progress. When the connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection using assigned property values.

The value of the **HostName** or **HostAddress** property will be used to determine the host name or address to connect to.

The value of the **RemotePort** property will be used to determine the port number to connect to.

The value of the **Protocol** property will specify which protocol is used to establish the connection.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Connect Overload List

SocketWrench.Connect Method (String, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

Return Value

A boolean value which specifies if the connection has been established. If the socket is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the socket is in non-blocking mode, a return value of **true** indicates that the socket has been successfully created and the connection is in progress. When the connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method will use the value of the Protocol property to determine which protocol is used to establish the connection. By default, the **socketStream** protocol will be used.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Connect Overload List

SocketWrench.Connect Method (String, Int32, SocketProtocol, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal protocol As SocketProtocol, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   SocketProtocol protocol,
   int timeout
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

protocol

One of the SocketProtocol enumeration values which specify the type of socket to be created.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking sockets.

Return Value

A boolean value which specifies if the connection has been established. If the socket is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the socket is in non-blocking mode, a return value of **true** indicates that the socket has been successfully created and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

When this method is called with **socketDatagram** as the specified protocol, it does not actually establish a connection. Instead, it simply establishes a default destination address and port that is used with subsequent calls to the **Read** and **Write** methods.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Connect Overload List

SocketWrench.Connect Method (String, Int32, SocketProtocol, Int32, SocketOptions)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal protocol As SocketProtocol, _
   ByVal timeout As Integer, _
   ByVal options As SocketOptions _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   SocketProtocol protocol,
   int timeout,
   SocketOptions options
);
```

Parameters

hostName
A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort
An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

protocol
One of the SocketProtocol enumeration values which specify the type of socket to be created.

timeout
An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking sockets.

options
One or more of the SocketOptions enumeration flags.

Return Value

A boolean value which specifies if the connection has been established. If the socket is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the socket is in non-blocking mode, a return value of **true** indicates that the socket has been successfully created and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

When this method is called with **socketDatagram** as the specified protocol, it does not actually establish a connection. Instead, it simply establishes a default destination address and port that is used with subsequent calls to the **Read** and **Write** methods.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Connect Overload List

SocketWrench.Connect Method (String, Int32, SocketProtocol, Int32, SocketOptions, String, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal protocol As SocketProtocol, _
   ByVal timeout As Integer, _
   ByVal options As SocketOptions, _
   ByVal localAddress As String, _
   ByVal localPort As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   SocketProtocol protocol,
   int timeout,
   SocketOptions options,
   string localAddress,
   int localPort
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

protocol

One of the SocketProtocol enumeration values which specify the type of socket to be created.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking sockets.

options

One or more of the SocketOptions enumeration flags.

localAddress

A string which specifies the local Internet address that the socket should be bound to. To bind to any valid network interface on the local system, specify the address 0.0.0.0. Applications should only specify a particular address if it is absolutely necessary. In most cases a local address is not required when establishing a client connection.

localPort

An integer value which specifies a local port number that the socket should be bound to. To bind to any available port number, specify a port number of 0. Applications should only specify a particular port number if it is absolutely necessary. The maximum valid port number is 65535.

Return Value

A boolean value which specifies if the connection has been established. If the socket is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the socket is in non-blocking mode, a return value of **true** indicates that the socket has been successfully created and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

When this method is called with **socketDatagram** as the specified protocol, it does not actually establish a connection. Instead, it simply establishes a default destination address and port that is used with subsequent calls to the Read and Write methods.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Connect Overload List

SocketWrench.Connect Method (String, Int32, Int32)

Establish a connection with a remote host.

```
[Visual Basic]
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer _
) As Boolean
```

```
[C#]
public bool Connect(
   string hostName,
   int hostPort,
   int timeout
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking sockets.

Return Value

A boolean value which specifies if the connection has been established. If the socket is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the socket is in non-blocking mode, a return value of **true** indicates that the socket has been successfully created and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method will use the value of the Protocol property to determine which protocol is used to establish the connection. By default, the **socketStream** protocol will be used.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Connect Overload List

SocketWrench.Disconnect Method

Terminate the connection with a remote host.

[Visual Basic]

```
Public Sub Disconnect()
```

[C#]

```
public void Disconnect();
```

Remarks

The **Disconnect** method terminates the connection with the remote host and closes the socket handle allocated by the class. Note that the socket is not immediately released when the connection is terminated and will enter a wait state for two minutes. After the time wait period has elapsed, the socket will be released by the operating system. This is a normal safety mechanism to handle any packets that may arrive after the connection has been closed.

To immediately terminate the connection and release the socket, use the **Abort** method.

See Also

SocketWrench Class | SocketTools Namespace | Abort Method

SocketWrench.Dispose Method

Releases all resources used by SocketWrench.

Overload List

Releases all resources used by SocketWrench.

public void Dispose();

Releases the unmanaged resources allocated by the SocketWrench class and optionally releases the managed resources.

protected virtual void Dispose(bool);

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Dispose Method ()

Releases all resources used by SocketWrench.

[Visual Basic]

```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]

```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Dispose Overload List

SocketWrench.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the SocketWrench class and optionally releases the managed resources.

[Visual Basic]

```
Overridable Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]

```
protected virtual void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **SocketWrench** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Dispose Overload List

SocketWrench.Finalize Method

Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.

[Visual Basic]

```
Overrides Protected Sub Finalize()
```

[C#]

```
protected override void Finalize();
```

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Initialize Method

Initialize an instance of the SocketWrench class.

Overload List

Initialize an instance of the SocketWrench class.

public bool Initialize();

Initialize an instance of the SocketWrench class.

public bool Initialize(string);

See Also

SocketWrench Class | SocketTools Namespace | Uninitialize Method

SocketWrench.Initialize Method ()

Initialize an instance of the SocketWrench class.

[Visual Basic]

```
Overloads Public Function Initialize() As Boolean
```

[C#]

```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the SocketWrench class, loading the networking library and allocating resources for the current thread. Typically it is not necessary to explicitly call this method because the instance of the class is initialized by the class constructor. However, if the **Uninitialize** method is called, the class must be re-initialized before any other methods are called.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Initialize Overload List | Uninitialize Method

SocketWrench.Initialize Method (String)

Initialize an instance of the SocketWrench class.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the SocketWrench class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of SocketWrench can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the SocketWrench class.

Example

The following example shows how to use the Initialize method to initialize an instance of the class. This example assumes that the license key string has been defined in code.

```
SocketTools.SocketWrench socket = new SocketTools.SocketWrench();

if (socket.Initialize(strLicenseKey) == false)
{
    MessageBox.Show(socket.LastErrorString, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    return;
}
```

```
Dim Socket As New SocketTools.SocketWrench

If Socket.Initialize(strLicenseKey) = False Then
    MsgBox(Socket.LastErrorString, vbIconExclamation)
    Exit Sub
End If
```

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

SocketWrench.Listen Method

Listen for incoming client connections.

Overload List

Listen for incoming client connections.

public bool Listen();

Listen for incoming client connections, specifying the local port number.

public bool Listen(int);

Listen for incoming client connections, specifying the local network address and port number.

public bool Listen(string,int);

Listen for incoming client connections, specifying the local network address, port number and connection backlog.

public bool Listen(string,int,int);

See Also

SocketWrench Class | SocketTools Namespace | Blocking Property | LocalAddress Property | LocalPort Property | Accept Method | OnAccept Event

SocketWrench.Listen Method ()

Listen for incoming client connections.

[Visual Basic]

```
Overloads Public Function Listen() As Boolean
```

[C#]

```
public bool Listen();
```

Return Value

A boolean value which specifies if the listening socket could be created successfully. A value of **true** indicates that a listening socket has been created. A value of **false** indicates that a listening socket could not be created using the specified address or port number and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The value of the **LocalAddress** property is used to specify the network address that will be used to listen for client connections. If the property has not been set, or is set to the address 0.0.0.0 then connections will be listened for on any valid network adapter configured on the system.

The value of the **LocalPort** property is used to specify the port number to listen for connections on.

After the listening socket has been created, the application should then call the **Accept** method to wait for a client to establish a connection. If the **Blocking** property is set to **false**, then the **OnAccept** event will fire when a client attempts to establish a connection.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Listen Overload List | Blocking Property | LocalAddress Property | LocalPort Property | Accept Method | OnAccept Event

SocketWrench.Listen Method (Int32)

Listen for incoming client connections, specifying the local port number.

[Visual Basic]

```
Overloads Public Function Listen( _
   ByVal localPort As Integer _
) As Boolean
```

[C#]

```
public bool Listen(
   int localPort
);
```

Parameters

localPort

An integer argument which specifies the port number to listen for connections on. The minimum port value is 1, the maximum port value is 65535.

Return Value

A boolean value which specifies if the listening socket could be created successfully. A value of **true** indicates that a listening socket has been created. A value of **false** indicates that a listening socket could not be created using the specified address or port number and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The value of the **LocalAddress** property is used to specify the network address that will be used to listen for client connections. If the property has not been set, or is set to the address 0.0.0.0 then connections will be listened for on any valid network adapter configured on the system.

After the listening socket has been created, the application should then call the **Accept** method to wait for a client to establish a connection. If the **Blocking** property is set to **false**, then the **OnAccept** event will fire when a client attempts to establish a connection.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Listen Overload List | Blocking Property | LocalAddress Property | Accept Method | OnAccept Event

SocketWrench.Listen Method (String, Int32)

Listen for incoming client connections, specifying the local network address and port number.

```
[Visual Basic]
Overloads Public Function Listen( _
   ByVal localAddress As String, _
   ByVal localPort As Integer _
) As Boolean
```

```
[C#]
public bool Listen(
   string localAddress,
   int localPort
);
```

Parameters

localAddress

A string argument which specifies the IP address of the network adapter that the class should use when listening for connection requests. If this argument is not specified, the class will bind to any suitable adapter on the local system. An address of 0.0.0.0 specifies that it should listen for connections on any network adapter configured on the system.

localPort

An integer argument which specifies the port number to listen for connections on. The minimum port value is 1, the maximum port value is 65535.

Return Value

A boolean value which specifies if the listening socket could be created successfully. A value of **true** indicates that a listening socket has been created. A value of **false** indicates that a listening socket could not be created using the specified address or port number and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

After the listening socket has been created, the application should then call the **Accept** method to wait for a client to establish a connection. If the **Blocking** property is set to **false**, then the **OnAccept** event will fire when a client attempts to establish a connection.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Listen Overload List | Blocking Property | Accept Method | OnAccept Event

SocketWrench.Listen Method (String, Int32, Int32)

Listen for incoming client connections, specifying the local network address, port number and connection backlog.

[Visual Basic]

```
Overloads Public Function Listen( _
   ByVal localAddress As String, _
   ByVal localPort As Integer, _
   ByVal backlog As Integer _
) As Boolean
```

[C#]

```
public bool Listen(
   string localAddress,
   int localPort,
   int backlog
);
```

Parameters

localAddress

A string argument which specifies the IP address of the network adapter that the class should use when listening for connection requests. If this argument is not specified, the class will bind to any suitable adapter on the local system. An address of 0.0.0.0 specifies that it should listen for connections on any network adapter configured on the system.

localPort

An integer argument which specifies the port number to listen for connections on. The minimum port value is 1, the maximum port value is 65535.

backlog

An integer argument which specifies the maximum size of the queue used to manage pending connections to the service. If the argument is set to value which exceeds the maximum size for the underlying service provider, it will be silently adjusted to the nearest legal value.

Return Value

A boolean value which specifies if the listening socket could be created successfully. A value of **true** indicates that a listening socket has been created. A value of **false** indicates that a listening socket could not be created using the specified address or port number and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

After the listening socket has been created, the application should then call the **Accept** method to wait for a client to establish a connection. If the **Blocking** property is set to **false**, then the **OnAccept** event will fire when a client attempts to establish a connection.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Listen Overload List | Blocking Property | Accept Method | OnAccept Event

SocketWrench.Read Method

Read data from the socket and store it in a byte array.

Overload List

Read data from the socket and store it in a byte array.

public int Read(byte[]);

Read data from the socket and store it in a byte array.

public int Read(byte[],int);

Read data from the socket and store it in a string.

public int Read(ref string);

Read data from the socket and store it in a string.

public int Read(ref string,int);

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Read Method (Byte[])

Read data from the socket and store it in a byte array.

[Visual Basic]

```
Overloads Public Function Read( _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Read(
   byte[] buffer
);
```

Parameters

buffer

A byte array that the data will be stored in.

Return Value

An integer value which specifies the number of bytes actually read from the socket. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the socket, up to the size of the byte array passed to the method. If no data is available to be read, an error will be generated if the socket is in non-blocking mode. If the socket is in blocking mode, the program will stop until data is received from the server or the connection is closed.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Read Overload List

SocketWrench.Read Method (Byte[], Int32)

Read data from the socket and store it in a byte array.

[Visual Basic]

```
Overloads Public Function Read( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Read(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array that the data will be stored in.

length

An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually read from the socket. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the socket, up to the number of bytes specified. If no data is available to be read, an error will be generated if the socket is in non-blocking mode. If the socket is in blocking mode, the program will stop until data is received from the server or the connection is closed.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Read Overload List

SocketWrench.Read Method (String)

Read data from the socket and store it in a string.

[Visual Basic]

```
Overloads Public Function Read( _
   ByRef buffer As String _
) As Integer
```

[C#]

```
public int Read(
   ref string buffer
);
```

Parameters

buffer

A string that will contain the data read from the socket.

Return Value

An integer value which specifies the number of bytes actually read from the socket. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the socket, up to a maximum of 8192 bytes. If no data is available to be read, an error will be generated if the socket is in non-blocking mode. If the socket is in blocking mode, the program will stop until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Read Overload List

SocketWrench.Read Method (String, Int32)

Read data from the socket and store it in a string.

[Visual Basic]

```
Overloads Public Function Read( _
   ByRef buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Read(
   ref string buffer,
   int length
);
```

Parameters

buffer

A string that will contain the data read from the socket.

length

An integer value which specifies the maximum number of bytes of data to read.

Return Value

An integer value which specifies the number of bytes actually read from the socket. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the socket, up to the number of bytes specified. If no data is available to be read, an error will be generated if the socket is in non-blocking mode. If the socket is in blocking mode, the program will stop until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Read Overload List

SocketWrench.ReadFrom Method

Read data from the socket and store it in a byte array.

Overload List

Read data from the socket and store it in a byte array.

public int ReadFrom(byte[],int,ref string,ref int);

Read data from the socket and store it in a byte array.

public int ReadFrom(byte[],ref string,ref int);

Read data from the socket and store it in a string.

public int ReadFrom(ref string,int,ref string,ref int);

Read data from the socket and store it in a string.

public int ReadFrom(ref string,ref string,ref int);

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.ReadFrom Method (Byte[], Int32, String, Int32)

Read data from the socket and store it in a byte array.

[Visual Basic]

```
Overloads Public Function ReadFrom( _
   ByVal buffer As Byte(), _
   ByVal length As Integer, _
   ByRef hostAddress As String, _
   ByRef hostPort As Integer _
) As Integer
```

[C#]

```
public int ReadFrom(
   byte[] buffer,
   int length,
   ref string hostAddress,
   ref int hostPort
);
```

Parameters

buffer

A byte array that the data will be stored in.

length

An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

hostAddress

A string passed by reference that will contain the remote host Internet address when the method returns. For stream sockets, this will be the same as the address used to establish the connection. For datagram sockets, this will specify the address of host that sent the datagram.

hostPort

An integer passed by reference that will contain the remote host port number when the method returns. For stream sockets, this will be the same as the port number used to establish the connection. For datagram sockets, this will specify the port number used by the host that sent the datagram.

Return Value

An integer value which specifies the number of bytes actually read from the socket. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **ReadFrom** method returns data that has been read from the socket, up to the number of bytes specified. If no data is available to be read, an error will be generated if the socket is in non-blocking mode. If the socket is in blocking mode, the program will stop until data is received from the server or the connection is closed.

This method is typically used when reading binary data from a datagram socket.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.ReadFrom Overload List

SocketWrench.ReadFrom Method (Byte[], String, Int32)

Read data from the socket and store it in a byte array.

[Visual Basic]

```
Overloads Public Function ReadFrom( _
   ByVal buffer As Byte(), _
   ByRef hostAddress As String, _
   ByRef hostPort As Integer _
) As Integer
```

[C#]

```
public int ReadFrom(
   byte[] buffer,
   ref string hostAddress,
   ref int hostPort
);
```

Parameters

buffer

A byte array that the data will be stored in.

hostAddress

A string passed by reference that will contain the remote host Internet address when the method returns. For stream sockets, this will be the same as the address used to establish the connection. For datagram sockets, this will specify the address of host that sent the datagram.

hostPort

An integer passed by reference that will contain the remote host port number when the method returns. For stream sockets, this will be the same as the port number used to establish the connection. For datagram sockets, this will specify the port number used by the host that sent the datagram.

Return Value

An integer value which specifies the number of bytes actually read from the socket. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **ReadFrom** method returns data that has been read from the socket, up to the number of bytes specified. If no data is available to be read, an error will be generated if the socket is in non-blocking mode. If the socket is in blocking mode, the program will stop until data is received from the server or the connection is closed.

This method is typically used when reading binary data from a datagram socket.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.ReadFrom Overload List

SocketWrench.ReadFrom Method (String, Int32, String, Int32)

Read data from the socket and store it in a string.

[Visual Basic]

```
Overloads Public Function ReadFrom( _
   ByRef buffer As String, _
   ByVal length As Integer, _
   ByRef hostAddress As String, _
   ByRef hostPort As Integer _
) As Integer
```

[C#]

```
public int ReadFrom(
   ref string buffer,
   int length,
   ref string hostAddress,
   ref int hostPort
);
```

Parameters

buffer
A string that will contain the data read from the socket.

length
An integer value which specifies the maximum number of bytes of data to read.

hostAddress
A string passed by reference that will contain the remote host Internet address when the method returns. For stream sockets, this will be the same as the address used to establish the connection. For datagram sockets, this will specify the address of host that sent the datagram.

hostPort
An integer passed by reference that will contain the remote host port number when the method returns. For stream sockets, this will be the same as the port number used to establish the connection. For datagram sockets, this will specify the port number used by the host that sent the datagram.

Return Value

An integer value which specifies the number of bytes actually read from the socket. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **ReadFrom** method returns data that has been read from the socket, up to the number of bytes specified. If no data is available to be read, an error will be generated if the socket is in non-blocking mode. If the socket is in blocking mode, the program will stop until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

This method is typically used when reading text data from a datagram socket.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.ReadFrom Overload List

SocketWrench.ReadFrom Method (String, String, Int32)

Read data from the socket and store it in a string.

[Visual Basic]

```
Overloads Public Function ReadFrom( _
   ByRef buffer As String, _
   ByRef hostAddress As String, _
   ByRef hostPort As Integer _
) As Integer
```

[C#]

```
public int ReadFrom(
   ref string buffer,
   ref string hostAddress,
   ref int hostPort
);
```

Parameters

buffer
A string that will contain the data read from the socket.

hostAddress
A string passed by reference that will contain the remote host Internet address when the method returns. For stream sockets, this will be the same as the address used to establish the connection. For datagram sockets, this will specify the address of host that sent the datagram.

hostPort
An integer passed by reference that will contain the remote host port number when the method returns. For stream sockets, this will be the same as the port number used to establish the connection. For datagram sockets, this will specify the port number used by the host that sent the datagram.

Return Value

An integer value which specifies the number of bytes actually read from the socket. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **ReadFrom** method returns data that has been read from the socket, up to the maximum size of a datagram. If no data is available to be read, an error will be generated if the socket is in non-blocking mode. If the socket is in blocking mode, the program will stop until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

This method is typically used when reading text data from a datagram socket.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.ReadFrom Overload List

SocketWrench.ReadLine Method

Read up to a line of data from the socket and return it in a string buffer.

Overload List

Read up to a line of data from the socket and return it in a string buffer.

public bool ReadLine(ref string);

Read up to a line of data from the socket and return it in a string buffer.

public bool ReadLine(ref string,int);

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.ReadLine Method (String)

Read up to a line of data from the socket and return it in a string buffer.

[Visual Basic]
```
Overloads Public Function ReadLine( _
   ByRef buffer As String _
) As Boolean
```

[C#]
```
public bool ReadLine(
   ref string buffer
);
```

Parameters

buffer

A string which will contain the data read from the socket.

Return Value

This method returns a Boolean value which specifies if a line of data has been read. A value of **true** indicates a line of data has been read. If an error occurs or there is no more data available to read, then the method will return **false**. It is possible for data to be returned in the string buffer even if the return value is **false**. Applications should check the length of the string after the method returns to determine if any data was copied into the buffer. For example, if a timeout occurs while the method is waiting for more data to arrive on the socket, it will return zero; however, data may have already been copied into the string buffer prior to the error condition. It is the responsibility of the application to process that data, regardless of the method return value.

Remarks

The **ReadLine** method reads data from the socket up to 8192 bytes in length or until an end-of-line character sequence is encountered. Unlike the **Read** method which reads arbitrary bytes of data, this method is specifically designed to return a single line of text data in a string variable. When an end-of-line character sequence is encountered, the method will stop and return the data up to that point; the string will not contain the carriage-return or linefeed characters.

There are some limitations when using the **ReadLine** method. The method should only be used to read text, never binary data. In particular, it will discard nulls, linefeed and carriage return control characters. This method will force the current thread to block until an end-of-line character sequence is processed, the read operation times out or the remote host closes its end of the socket connection. If the **Blocking** property is set to **false**, calling this method will automatically switch the socket into a blocking mode, read the data and then restore the socket to non-blocking mode. If another socket operation is attempted while **ReadLine** is blocked waiting for data from the remote host, an error will occur. It is recommended that this method only be used with blocking socket connections.

The **Read** and **ReadLine** methods can be intermixed, however be aware that the **Read** method will consume any data that has already been buffered by the **ReadLine** method and this may have unexpected results.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.ReadLine Overload List

SocketWrench.ReadLine Method (String, Int32)

Read up to a line of data from the socket and return it in a string buffer.

[Visual Basic]

```
Overloads Public Function ReadLine( _
   ByRef buffer As String, _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool ReadLine(
   ref string buffer,
   int length
);
```

Parameters

buffer

A string which will contain the data read from the socket.

length

An integer value which specifies the maximum number of bytes of data to read.

Return Value

This method returns a Boolean value which specifies if a line of data has been read. A value of **true** indicates a line of data has been read. If an error occurs or there is no more data available to read, then the method will return **false**. It is possible for data to be returned in the string buffer even if the return value is **false**. Applications should check the length of the string after the method returns to determine if any data was copied into the buffer. For example, if a timeout occurs while the method is waiting for more data to arrive on the socket, it will return zero; however, data may have already been copied into the string buffer prior to the error condition. It is the responsibility of the application to process that data, regardless of the method return value.

Remarks

The **ReadLine** method reads data from the socket up to the specified number of bytes or until an end-of-line character sequence is encountered. Unlike the **Read** method which reads arbitrary bytes of data, this method is specifically designed to return a single line of text data in a string variable. When an end-of-line character sequence is encountered, the method will stop and return the data up to that point; the string will not contain the carriage-return or linefeed characters.

There are some limitations when using the **ReadLine** method. The method should only be used to read text, never binary data. In particular, it will discard nulls, linefeed and carriage return control characters. This method will force the current thread to block until an end-of-line character sequence is processed, the read operation times out or the remote host closes its end of the socket connection. If the **Blocking** property is set to **false**, calling this method will automatically switch the socket into a blocking mode, read the data and then restore the socket to non-blocking mode. If another socket operation is attempted while **ReadLine** is blocked waiting for data from the remote host, an error will occur. It is recommended that this method only be used with blocking socket connections.

The **Read** and **ReadLine** methods can be intermixed, however be aware that the **Read** method will consume any data that has already been buffered by the **ReadLine** method and this may have unexpected results.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.ReadLine Overload List

SocketWrench.ReadStream Method

Read a data stream from the socket and store it in the specified byte array.

Overload List

Read a data stream from the socket and store it in the specified byte array.

public bool ReadStream(byte[],ref int);

Read a data stream from the socket and store it in the specified byte array.

public bool ReadStream(byte[],ref int,byte[]);

Read a data stream from the socket and store it in the specified byte array.

public bool ReadStream(byte[],ref int,byte[],SocketStream);

Read a data stream from the socket and store it in the specified string.

public bool ReadStream(ref string);

Read a data stream from the socket and store it in the specified string.

public bool ReadStream(ref string,bool);

Read a data stream from the socket and store it in the specified string.

public bool ReadStream(ref string,ref int);

Read a data stream from the socket and store it in the specified string.

public bool ReadStream(ref string,ref int,bool);

Read a data stream from the socket and store it in the specified string.

public bool ReadStream(ref string,ref int,string,bool);

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.ReadStream Method (Byte[], Int32)

Read a data stream from the socket and store it in the specified byte array.

[Visual Basic]

```
Overloads Public Function ReadStream( _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool ReadStream(
   byte[] buffer,
   ref int length
);
```

Parameters

buffer

A byte array that the data will be stored in.

length

An integer value passed by reference which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller. When the method returns, this value will be updated with the actual number of bytes read from the socket.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method will force the current thread to block until the operation completes. If this method is called and the **Blocking** property is set to **false**, it will automatically switch the socket into a blocking mode, read the data stream and then restore the socket to non-blocking mode when it has finished. If another socket operation is attempted while **ReadStream** is blocked waiting for data from the remote host, an error will occur. It is recommended that this method only be used with blocking (synchronous) socket connections; if the application needs to establish multiple simultaneous connections, it should create worker threads to manage each connection.

It is possible for data to be returned in the buffer even if the method returns **false**. Applications should also check the value of the ***length*** parameter to determine if any data was copied into the buffer. For example, if a timeout occurs while the method is waiting for more data to arrive on the socket, it will return zero; however, data may have already been copied into the buffer prior to the error condition. It is the responsibility of the application to process that data, regardless of the method return value.

Because **ReadStream** can potentially cause the current thread to block for long periods of time as the data stream is being read, the class will periodically generate **OnProgress** events. An application can use this event to update the user interface as the data is being read. Note that an application should never perform a blocking operation inside the event handler.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.ReadStream Overload List

SocketWrench.ReadStream Method (Byte[], Int32, Byte[])

Read a data stream from the socket and store it in the specified byte array.

[Visual Basic]

```
Overloads Public Function ReadStream( _
   ByVal buffer As Byte(), _
   ByRef length As Integer, _
   ByVal marker As Byte() _
) As Boolean
```

[C#]

```
public bool ReadStream(
   byte[] buffer,
   ref int length,
   byte[] marker
);
```

Parameters

buffer

A byte array that the data will be stored in.

length

An integer value passed by reference which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller. When the method returns, this value will be updated with the actual number of bytes read from the socket.

marker

An array of bytes which is used to designate the logical end of the data stream. When this byte sequence is encountered by the method, it will stop reading and return to the caller. The buffer will contain all of the data read from the socket up to and including the end-of-stream marker.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method will force the current thread to block until the operation completes. If this method is called and the **Blocking** property is set to **false**, it will automatically switch the socket into a blocking mode, read the data stream and then restore the socket to non-blocking mode when it has finished. If another socket operation is attempted while **ReadStream** is blocked waiting for data from the remote host, an error will occur. It is recommended that this method only be used with blocking (synchronous) socket connections; if the application needs to establish multiple simultaneous connections, it should create worker threads to manage each connection.

It is possible for data to be returned in the buffer even if the method returns **false**. Applications should also check the value of the *length* parameter to determine if any data was copied into the buffer. For example, if a timeout occurs while the method is waiting for more data to arrive on the socket, it will return zero; however, data may have already been copied into the buffer prior to the error condition. It is the responsibility of the application to process that data, regardless of the method return value.

Because **ReadStream** can potentially cause the current thread to block for long periods of time as the data stream is being read, the class will periodically generate **OnProgress** events. An application can use this event to update the user interface as the data is being read. Note that an application should never perform a blocking operation inside the event handler.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.ReadStream Overload List

SocketWrench.ReadStream Method (Byte[], Int32, Byte[], SocketStream)

Read a data stream from the socket and store it in the specified byte array.

[Visual Basic]

```
Overloads Public Function ReadStream( _
   ByVal buffer As Byte(), _
   ByRef length As Integer, _
   ByVal marker As Byte(), _
   ByVal options As SocketStream _
) As Boolean
```

[C#]

```
public bool ReadStream(
   byte[] buffer,
   ref int length,
   byte[] marker,
   SocketStream options
);
```

Parameters

buffer

A byte array that the data will be stored in.

length

An integer value passed by reference which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller. When the method returns, this value will be updated with the actual number of bytes read from the socket.

marker

An array of bytes which is used to designate the logical end of the data stream. When this byte sequence is encountered by the method, it will stop reading and return to the caller. The buffer will contain all of the data read from the socket up to and including the end-of-stream marker.

options

One of the SocketStream enumeration values which specifies how the data is processed.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method will force the current thread to block until the operation completes. If this method is called and the **Blocking** property is set to **false**, it will automatically switch the socket into a blocking mode, read the data stream and then restore the socket to non-blocking mode when it has finished. If another socket operation is attempted while **ReadStream** is blocked waiting for data from the remote host, an error will occur. It is recommended that this method only be used with blocking (synchronous) socket connections; if the application needs to establish multiple simultaneous connections, it should create worker threads to manage each connection.

It is possible for data to be returned in the buffer even if the method returns **false**. Applications should also check the value of the *length* parameter to determine if any data was copied into the buffer. For example, if a timeout occurs while the method is waiting for more data to arrive on the socket, it will return zero; however, data may have already been copied into the buffer prior to the error condition. It is the responsibility of the application to process that data, regardless of the method return value.

Because **ReadStream** can potentially cause the current thread to block for long periods of time as the data stream is being read, the class will periodically generate **OnProgress** events. An application can use this event to update the user interface as the data is being read. Note that an application should never perform a blocking operation inside the event handler.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.ReadStream Overload List

SocketWrench.ReadStream Method (String)

Read a data stream from the socket and store it in the specified string.

[Visual Basic]

```
Overloads Public Function ReadStream( _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool ReadStream(
   ref string buffer
);
```

Parameters

buffer

A string that will contain the data read from the socket.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method will force the current thread to block until the operation completes. If this method is called and the **Blocking** property is set to **false**, it will automatically switch the socket into a blocking mode, read the data stream and then restore the socket to non-blocking mode when it has finished. If another socket operation is attempted while **ReadStream** is blocked waiting for data from the remote host, an error will occur. It is recommended that this method only be used with blocking (synchronous) socket connections; if the application needs to establish multiple simultaneous connections, it should create worker threads to manage each connection.

It is possible for data to be returned in the buffer even if the method returns **false**. Applications should also check the length of the string to determine if any data was copied into the buffer. For example, if a timeout occurs while the method is waiting for more data to arrive on the socket, it will return zero; however, data may have already been copied into the buffer prior to the error condition. It is the responsibility of the application to process that data, regardless of the method return value.

Because **ReadStream** can potentially cause the current thread to block for long periods of time as the data stream is being read, the class will periodically generate **OnProgress** events. An application can use this event to update the user interface as the data is being read. Note that an application should never perform a blocking operation inside the event handler.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.ReadStream Overload List

SocketWrench.ReadStream Method (String, Boolean)

Read a data stream from the socket and store it in the specified string.

[Visual Basic]

```
Overloads Public Function ReadStream( _
   ByRef buffer As String, _
   ByVal convertText As Boolean _
) As Boolean
```

[C#]

```
public bool ReadStream(
   ref string buffer,
   bool convertText
);
```

Parameters

buffer

A string that will contain the data read from the socket.

convertText

A boolean flag which specifies if the data data stream is considered to be textual and should be modified so that end-of-line character sequences are converted to follow standard Windows conventions. This will ensure that all lines of text are terminated with a carriage-return and linefeed sequence. Because this option modifies the data stream, it should never be used with binary data. Using this option may result in the amount of data returned in the buffer to be larger than the source data. For example, if the source data only terminates a line of text with a single linefeed, this option will have the effect of inserting a carriage-return character before each linefeed.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method will force the current thread to block until the operation completes. If this method is called and the **Blocking** property is set to **false**, it will automatically switch the socket into a blocking mode, read the data stream and then restore the socket to non-blocking mode when it has finished. If another socket operation is attempted while **ReadStream** is blocked waiting for data from the remote host, an error will occur. It is recommended that this method only be used with blocking (synchronous) socket connections; if the application needs to establish multiple simultaneous connections, it should create worker threads to manage each connection.

It is possible for data to be returned in the buffer even if the method returns **false**. Applications should also check the length of the string to determine if any data was copied into the buffer. For example, if a timeout occurs while the method is waiting for more data to arrive on the socket, it will return zero; however, data may have already been copied into the buffer prior to the error condition. It is the responsibility of the application to process that data, regardless of the method return value.

Because **ReadStream** can potentially cause the current thread to block for long periods of time as the data stream is being read, the class will periodically generate **OnProgress** events. An application can use this event to update the user interface as the data is being read. Note that an application should never perform a blocking operation inside the event handler.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.ReadStream Overload List

SocketWrench.ReadStream Method (String, Int32)

Read a data stream from the socket and store it in the specified string.

[Visual Basic]

```
Overloads Public Function ReadStream( _
   ByRef buffer As String, _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool ReadStream(
   ref string buffer,
   ref int length
);
```

Parameters

buffer

A string that will contain the data read from the socket.

length

An integer value passed by reference which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller. When the method returns, this value will be updated with the actual number of bytes read from the socket.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method will force the current thread to block until the operation completes. If this method is called and the **Blocking** property is set to **false**, it will automatically switch the socket into a blocking mode, read the data stream and then restore the socket to non-blocking mode when it has finished. If another socket operation is attempted while **ReadStream** is blocked waiting for data from the remote host, an error will occur. It is recommended that this method only be used with blocking (synchronous) socket connections; if the application needs to establish multiple simultaneous connections, it should create worker threads to manage each connection.

It is possible for data to be returned in the buffer even if the method returns **false**. Applications should also check the value of the ***length*** parameter to determine if any data was copied into the buffer. For example, if a timeout occurs while the method is waiting for more data to arrive on the socket, it will return zero; however, data may have already been copied into the buffer prior to the error condition. It is the responsibility of the application to process that data, regardless of the method return value.

Because **ReadStream** can potentially cause the current thread to block for long periods of time as the data stream is being read, the class will periodically generate **OnProgress** events. An application can use this event to update the user interface as the data is being read. Note that an application should never perform a blocking operation inside the event handler.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.ReadStream Overload List

SocketWrench.ReadStream Method (String, Int32, Boolean)

Read a data stream from the socket and store it in the specified string.

[Visual Basic]

```
Overloads Public Function ReadStream( _
   ByRef buffer As String, _
   ByRef length As Integer, _
   ByVal convertText As Boolean _
) As Boolean
```

[C#]

```
public bool ReadStream(
   ref string buffer,
   ref int length,
   bool convertText
);
```

Parameters

buffer
: A string that will contain the data read from the socket.

length
: An integer value passed by reference which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller. When the method returns, this value will be updated with the actual number of bytes read from the socket.

convertText
: A boolean flag which specifies if the data data stream is considered to be textual and should be modified so that end-of-line character sequences are converted to follow standard Windows conventions. This will ensure that all lines of text are terminated with a carriage-return and linefeed sequence. Because this option modifies the data stream, it should never be used with binary data. Using this option may result in the amount of data returned in the buffer to be larger than the source data. For example, if the source data only terminates a line of text with a single linefeed, this option will have the effect of inserting a carriage-return character before each linefeed.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method will force the current thread to block until the operation completes. If this method is called and the **Blocking** property is set to **false**, it will automatically switch the socket into a blocking mode, read the data stream and then restore the socket to non-blocking mode when it has finished. If another socket operation is attempted while **ReadStream** is blocked waiting for data from the remote host, an error will occur. It is recommended that this method only be used with blocking (synchronous) socket connections; if the application needs to establish multiple simultaneous connections, it should create worker threads to manage each connection.

It is possible for data to be returned in the buffer even if the method returns **false**. Applications should also check the value of the ***length*** parameter to determine if any data was copied into the buffer. For example, if a timeout occurs while the method is waiting for more data to arrive on the socket, it will return zero; however, data may have already been copied into the buffer prior to the error condition. It is the responsibility of the application to process that data, regardless of the method return value.

Because **ReadStream** can potentially cause the current thread to block for long periods of time as the data stream is being read, the class will periodically generate **OnProgress** events. An application can use this event to update the user interface as the data is being read. Note that an application should never perform a blocking operation inside the event handler.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.ReadStream Overload List

SocketWrench.ReadStream Method (String, Int32, String, Boolean)

Read a data stream from the socket and store it in the specified string.

[Visual Basic]

```
Overloads Public Function ReadStream( _
   ByRef buffer As String, _
   ByRef length As Integer, _
   ByVal marker As String, _
   ByVal convertText As Boolean _
) As Boolean
```

[C#]

```
public bool ReadStream(
   ref string buffer,
   ref int length,
   string marker,
   bool convertText
);
```

Parameters

buffer
A string that will contain the data read from the socket.

length
An integer value passed by reference which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller. When the method returns, this value will be updated with the actual number of bytes read from the socket.

marker
A string which is used to designate the logical end of the data stream. When this character sequence is encountered by the method, it will stop reading and return to the caller. The string buffer will contain all of the data read from the socket up to and including the end-of-stream marker.

convertText
A boolean flag which specifies if the data data stream is considered to be textual and should be modified so that end-of-line character sequences are converted to follow standard Windows conventions. This will ensure that all lines of text are terminated with a carriage-return and linefeed sequence. Because this option modifies the data stream, it should never be used with binary data. Using this option may result in the amount of data returned in the buffer to be larger than the source data. For example, if the source data only terminates a line of text with a single linefeed, this option will have the effect of inserting a carriage-return character before each linefeed.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

This method will force the current thread to block until the operation completes. If this method is called and the **Blocking** property is set to **false**, it will automatically switch the socket into a blocking mode, read the data stream and then restore the socket to non-blocking mode when it has finished. If another socket operation is attempted while **ReadStream** is blocked waiting for data from the remote host, an error will occur. It is recommended that this method only be used with blocking (synchronous) socket connections; if the application needs to establish multiple simultaneous connections, it should create worker threads to manage each connection.

It is possible for data to be returned in the buffer even if the method returns **false**. Applications should also check the value of the ***length*** parameter to determine if any data was copied into the buffer. For example, if a timeout occurs while the method is waiting for more data to arrive on the socket, it will return zero; however, data may have already been copied into the buffer prior to the error condition. It is the responsibility of the application to process that data, regardless of the method return value.

Because **ReadStream** can potentially cause the current thread to block for long periods of time as the data stream is being read, the class will periodically generate **OnProgress** events. An application can use this event to update the user interface as the data is being read. Note that an application should never perform a blocking operation inside the event handler.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.ReadStream Overload List

SocketWrench.Reject Method

Rejects a connection request from a remote host.

[Visual Basic]

```
Public Function Reject() As Boolean
```

[C#]

```
public bool Reject();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Reject** method rejects a pending client connection and the remote host will see this as the connection being aborted. If there are no pending client connections at the time, this method will immediately return with an error indicating that the operation would cause the thread to block.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. If a socket has been allocated, it will be released and any active connections will be terminated. All properties will be reset to their default values.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Resolve Method

Resolves a host name to a host IP address.

[Visual Basic]

```
Public Function Resolve( _
   ByVal hostName As String, _
   ByRef hostAddress As String _
) As Boolean
```

[C#]

```
public bool Resolve(
   string hostName,
   ref string hostAddress
);
```

Parameters

hostName

A string which specifies the host name to be resolved.

hostAddress

A string which will contain the Internet address for the specified host.

Return Value

This method returns a Boolean value. If the host name can be resolved, the return value is **true**. If the host name cannot be resolved, the return value is **false**. To get extended error information, check the value of the **LastError** property.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Shutdown Method

Disable sending data on the socket.

Overload List

Disable sending data on the socket.

public bool Shutdown();

Disable sending or receiving data on the socket.

public bool Shutdown(ShutdownOptions);

See Also

SocketWrench Class | SocketTools Namespace | Disconnect Method

SocketWrench.Shutdown Method ()

Disable sending data on the socket.

[Visual Basic]
```
Overloads Public Function Shutdown() As Boolean
```

[C#]
```
public bool Shutdown();
```

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

In some asynchronous applications, it may be desirable for a client to inform the server that no further communication is wanted, while allowing the client to read any residual data that may reside in internal buffers on the client side. **Shutdown** accomplishes this because the socket handle is still valid after it has been called, although some or all communication with the remote host has ceased. Note that most applications do not typically need to use this method. To close a socket connection gracefully, you should use the **Disconnect** method.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Shutdown Overload List | Disconnect Method

SocketWrench.Shutdown Method (ShutdownOptions)

Disable sending or receiving data on the socket.

[Visual Basic]

```
Overloads Public Function Shutdown( _
   ByVal options As ShutdownOptions _
) As Boolean
```

[C#]

```
public bool Shutdown(
   ShutdownOptions options
);
```

Parameters

options

One of the ShutdownOptions enumeration values which specifies the operation that will no longer be allowed.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

In some asynchronous applications, it may be desirable for a client to inform the server that no further communication is wanted, while allowing the client to read any residual data that may reside in internal buffers on the client side. **Shutdown** accomplishes this because the socket handle is still valid after it has been called, although some or all communication with the remote host has ceased.

Note that most applications do not typically need to use this method. To close a socket connection gracefully, you should use the **Disconnect** method.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Shutdown Overload List | Disconnect Method

SocketWrench.StoreStream Method

Reads a data stream from the socket and stores it in the specified file.

Overload List

Reads a data stream from the socket and stores it in the specified file.

public bool StoreStream(string);

Reads a data stream from the socket and stores it in the specified file.

public bool StoreStream(string,ref int);

Reads a data stream from the socket and stores it in the specified file.

public bool StoreStream(string,ref int,bool);

Reads a data stream from the socket and stores it in the specified file.

public bool StoreStream(string,ref int,int,SocketStream);

Reads a data stream from the socket and stores it in the specified file.

public bool StoreStream(string,ref int,int,bool);

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.StoreStream Method (String)

Reads a data stream from the socket and stores it in the specified file.

[Visual Basic]

```
Overloads Public Function StoreStream( _
   ByVal fileName As String _
) As Boolean
```

[C#]

```
public bool StoreStream(
   string fileName
);
```

Parameters

fileName

A string variable that specifies the name of the file that will contain the data read from the socket. If the file does not exist, it will be created. If the file does exist, the contents will be overwritten.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

This method will force the current thread to block until the operation completes. If this method is called with the **Blocking** property set to **false**, it will automatically switch the socket into a blocking mode, read the data stream and then restore the socket to non-blocking mode when it has finished. If another socket operation is attempted while **StoreStream** is blocked waiting for data from the remote host, an error will occur. It is recommended that this method only be used with blocking (synchronous) socket connections; if the application needs to establish multiple simultaneous connections, it should create worker threads to manage each connection.

Because **StoreStream** can potentially cause the current thread to block for long periods of time as the data stream is being read, the class instance will periodically generate **OnProgress** events. An application can use this event to update the user interface as the data is being read. Note that an application should never perform a blocking operation inside the event handler.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.StoreStream Overload List

SocketWrench.StoreStream Method (String, Int32)

Reads a data stream from the socket and stores it in the specified file.

[Visual Basic]

```
Overloads Public Function StoreStream( _
   ByVal fileName As String, _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool StoreStream(
   string fileName,
   ref int length
);
```

Parameters

fileName

A string variable that specifies the name of the file that will contain the data read from the socket. If the file does not exist, it will be created. If the file does exist, the contents will be overwritten.

length

An integer value which specifies the maximum amount of data to be read from the socket. When the method returns, this variable will be updated with the actual number of bytes read. Note that because this argument is passed by reference and modified by the method, you must provide a variable, not a numeric constant. If the value is initialized to zero, this method will read data from the socket until the remote host disconnects or an error occurs.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

This method will force the current thread to block until the operation completes. If this method is called with the **Blocking** property set to **false**, it will automatically switch the socket into a blocking mode, read the data stream and then restore the socket to non-blocking mode when it has finished. If another socket operation is attempted while **StoreStream** is blocked waiting for data from the remote host, an error will occur. It is recommended that this method only be used with blocking (synchronous) socket connections; if the application needs to establish multiple simultaneous connections, it should create worker threads to manage each connection.

Because **StoreStream** can potentially cause the current thread to block for long periods of time as the data stream is being read, the class instance will periodically generate **OnProgress** events. An application can use this event to update the user interface as the data is being read. Note that an application should never perform a blocking operation inside the event handler.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.StoreStream Overload List

SocketWrench.StoreStream Method (String, Int32, Boolean)

Reads a data stream from the socket and stores it in the specified file.

[Visual Basic]

```
Overloads Public Function StoreStream( _
   ByVal fileName As String, _
   ByRef length As Integer, _
   ByVal convertText As Boolean _
) As Boolean
```

[C#]

```
public bool StoreStream(
   string fileName,
   ref int length,
   bool convertText
);
```

Parameters

fileName

A string variable that specifies the name of the file that will contain the data read from the socket. If the file does not exist, it will be created. If the file does exist, the contents will be overwritten.

length

An integer value which specifies the maximum amount of data to be read from the socket. When the method returns, this variable will be updated with the actual number of bytes read. Note that because this argument is passed by reference and modified by the method, you must provide a variable, not a numeric constant. If the value is initialized to zero, this method will read data from the socket until the remote host disconnects or an error occurs.

convertText

A boolean flag which specifies if the data data stream is considered to be textual and should be modified so that end-of-line character sequences are converted to follow standard Windows conventions. This will ensure that all lines of text are terminated with a carriage-return and linefeed sequence. Because this option modifies the data stream, it should never be used with binary data. Using this option may result in the amount of data stored in the file to be larger than the source data. For example, if the source data only terminates a line of text with a single linefeed, this option will have the effect of inserting a carriage-return character before each linefeed.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

This method will force the current thread to block until the operation completes. If this method is called with the **Blocking** property set to **false**, it will automatically switch the socket into a blocking mode, read the data stream and then restore the socket to non-blocking mode when it has finished. If another socket operation is attempted while **StoreStream** is blocked waiting for data from the remote host, an error will occur. It is recommended that this method only be used with blocking (synchronous) socket connections; if the application needs to establish multiple simultaneous connections, it should create worker threads to manage each connection.

Because **StoreStream** can potentially cause the current thread to block for long periods of time as the data stream is being read, the class instance will periodically generate **OnProgress** events. An application can use this event to update the user interface as the data is being read. Note that an application should never perform a blocking operation inside the event handler.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.StoreStream Overload List

SocketWrench.StoreStream Method (String, Int32, Int32, SocketStream)

Reads a data stream from the socket and stores it in the specified file.

[Visual Basic]

```
Overloads Public Function StoreStream( _
   ByVal fileName As String, _
   ByRef length As Integer, _
   ByVal offset As Integer, _
   ByVal options As SocketStream _
) As Boolean
```

[C#]

```
public bool StoreStream(
   string fileName,
   ref int length,
   int offset,
   SocketStream options
);
```

Parameters

fileName

A string variable that specifies the name of the file that will contain the data read from the socket. If the file does not exist, it will be created. If the file does exist, the contents will be overwritten.

length

An integer value which specifies the maximum amount of data to be read from the socket. When the method returns, this variable will be updated with the actual number of bytes read. Note that because this argument is passed by reference and modified by the method, you must provide a variable, not a numeric constant. If the value is initialized to zero, this method will read data from the socket until the remote host disconnects or an error occurs.

offset

A numeric value which specifies the byte offset into the file where the method will start storing data read from the socket. Note that all data after this offset will be truncated. If a value of zero is specified, the file will be completely overwritten if it already exists.

options

One of the SocketStream enumeration values which specifies how the data is processed.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

This method will force the current thread to block until the operation completes. If this method is called with the **Blocking** property set to **false**, it will automatically switch the socket into a blocking mode, read the data stream and then restore the socket to non-blocking mode when it has finished. If another socket operation is attempted while **StoreStream** is blocked waiting for data from the remote host, an error will occur. It is recommended that this method only be used with blocking (synchronous) socket connections; if the application needs to establish multiple simultaneous connections, it should create worker threads to manage each connection.

Because **StoreStream** can potentially cause the current thread to block for long periods of time as the data stream is being read, the class will periodically generate **OnProgress** events. An application can use this event to update the user interface as the data is being read. Note that an application should never perform a blocking operation inside the event handler.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.StoreStream Overload List

SocketWrench.StoreStream Method (String, Int32, Int32, Boolean)

Reads a data stream from the socket and stores it in the specified file.

[Visual Basic]

```
Overloads Public Function StoreStream( _
   ByVal fileName As String, _
   ByRef length As Integer, _
   ByVal offset As Integer, _
   ByVal convertText As Boolean _
) As Boolean
```

[C#]

```
public bool StoreStream(
   string fileName,
   ref int length,
   int offset,
   bool convertText
);
```

Parameters

fileName

A string variable that specifies the name of the file that will contain the data read from the socket. If the file does not exist, it will be created. If the file does exist, the contents will be overwritten.

length

An integer value which specifies the maximum amount of data to be read from the socket. When the method returns, this variable will be updated with the actual number of bytes read. Note that because this argument is passed by reference and modified by the method, you must provide a variable, not a numeric constant. If the value is initialized to zero, this method will read data from the socket until the remote host disconnects or an error occurs.

offset

A numeric value which specifies the byte offset into the file where the method will start storing data read from the socket. Note that all data after this offset will be truncated. If a value of zero is specified, the file will be completely overwritten if it already exists.

convertText

A boolean flag which specifies if the data data stream is considered to be textual and should be modified so that end-of-line character sequences are converted to follow standard Windows conventions. This will ensure that all lines of text are terminated with a carriage-return and linefeed sequence. Because this option modifies the data stream, it should never be used with binary data. Using this option may result in the amount of data stored in the file to be larger than the source data. For example, if the source data only terminates a line of text with a single linefeed, this option will have the effect of inserting a carriage-return character before each linefeed.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

This method will force the current thread to block until the operation completes. If this method is called with the **Blocking** property set to **false**, it will automatically switch the socket into a blocking mode, read the data stream and then restore the socket to non-blocking mode when it has finished. If another socket operation is attempted while **StoreStream** is blocked waiting for data from the remote host, an error will occur. It is recommended that this method only be used with blocking (synchronous) socket connections; if the application needs to establish multiple simultaneous connections, it should create worker threads to manage each connection.

Because **StoreStream** can potentially cause the current thread to block for long periods of time as the data stream is being read, the class instance will periodically generate **OnProgress** events. An application can use this event to update the user interface as the data is being read. Note that an application should never perform a blocking operation inside the event handler.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.StoreStream Overload List

SocketWrench.Uninitialize Method

Uninitialize the class library and release any resources allocated for the current thread.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method terminates any active connection, releases resources allocated for the current thread and unloads the networking library. After this method has been called, no further socket operations may be performed until the class instance has been re-initialized.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the class is no longer needed.

See Also

SocketWrench Class | SocketTools Namespace | Initialize Method

SocketWrench.Write Method

Write one or more bytes of data to the socket.

Overload List

Write one or more bytes of data to the socket.

public int Write(byte[]);

Write one or more bytes of data to the socket.

public int Write(byte[],int);

Write a string of characters to the socket.

public int Write(string);

Write a string of characters to the socket.

public int Write(string,int);

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.Write Method (Byte[])

Write one or more bytes of data to the socket.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Write(
   byte[] buffer
);
```

Parameters

buffer

A byte array that contains the data to be written to the socket.

Return Value

An integer value which specifies the number of bytes actually written to the socket. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to the remote host. If there is enough room in the socket's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the socket is in blocking mode, then the method will block until the data can be sent. If the socket is in non-blocking mode and the send buffer is full, an error will occur.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Write Overload List

SocketWrench.Write Method (Byte[], Int32)

Write one or more bytes of data to the socket.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Write(
   byte[] buffer,
   int length
);
```

Parameters

buffer
A byte array that contains the data to be written to the socket.

length
An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually written to the socket. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to the remote host. If there is enough room in the socket's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the socket is in blocking mode, then the method will block until the data can be sent. If the socket is in non-blocking mode and the send buffer is full, an error will occur.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Write Overload List

SocketWrench.Write Method (String)

Write a string of characters to the socket.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String _
) As Integer
```

[C#]

```
public int Write(
   string buffer
);
```

Parameters

buffer

A string which contains the data to be written to the socket.

Return Value

An integer value which specifies the number of characters actually written to the socket. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to the remote host. If there is enough room in the socket's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the socket is in blocking mode, then the method will block until the data can be sent. If the socket is in non-blocking mode and the send buffer is full, an error will occur.

The string will be converted to an array of bytes before being written to the socket. By default, the character encoding used will be for the current locale. Depending on the contents of the string, the number of bytes written may be different than the string length specified. This is because the conversion from Unicode to a byte array may result in a multi-byte character sequence.

You should never use strings to read and write binary data. Always use byte arrays to ensure that no character conversion is performed.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Write Overload List

SocketWrench.Write Method (String, Int32)

Write a string of characters to the socket.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Write(
   string buffer,
   int length
);
```

Parameters

buffer

A string which contains the data to be written to the socket.

length

An integer value which specifies the maximum number of characters to write. This value cannot be larger than the length of the string specified by the caller.

Return Value

An integer value which specifies the number of characters actually written to the socket. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to the remote host. If there is enough room in the socket's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the socket is in blocking mode, then the method will block until the data can be sent. If the socket is in non-blocking mode and the send buffer is full, an error will occur.

The string will be converted to an array of bytes before being written to the socket. By default, the character encoding used will be for the current locale. Depending on the contents of the string, the number of bytes written may be different than the string length specified. This is because the conversion from Unicode to a byte array may result in a multi-byte character sequence.

You should never use strings to read and write binary data. Always use byte arrays to ensure that no character conversion is performed.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.Write Overload List

SocketWrench.WriteLine Method

Send an empty line of text to the remote host, terminated by a carriage-return and linefeed.

Overload List

Send an empty line of text to the remote host, terminated by a carriage-return and linefeed.

public bool WriteLine();

Send a line of text to the remote host, terminated by a carriage-return and linefeed.

public bool WriteLine(string);

Send a line of text to the remote host, terminated by a carriage-return and linefeed.

public bool WriteLine(string,ref int);

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.WriteLine Method ()

Send an empty line of text to the remote host, terminated by a carriage-return and linefeed.

[Visual Basic]

```
Overloads Public Function WriteLine() As Boolean
```

[C#]

```
public bool WriteLine();
```

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

The **WriteLine** method will send an empty line of text, terminated by a carriage-return and linefeed. Calling this method will force the application to block until the complete line of text has been written, the write operation times out or the remote host aborts the connection. If this method is called with the **Blocking** property set to **false**, it will automatically switch the socket into a blocking mode, send the data and then restore the socket to non-blocking mode. If another socket operation is attempted while the **WriteLine** method is blocked sending data to the remote host, an error will occur. It is recommended that this method only be used with blocking socket connections.

The **Write** and **WriteLine** methods can be safely intermixed.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.WriteLine Overload List

SocketWrench.WriteLine Method (String)

Send a line of text to the remote host, terminated by a carriage-return and linefeed.

[Visual Basic]

```
Overloads Public Function WriteLine( _
   ByVal buffer As String _
) As Boolean
```

[C#]

```
public bool WriteLine(
   string buffer
);
```

Parameters

buffer

A string which contains the data that will be sent to the remote host. The data will always be terminated with a carriage-return and linefeed control character sequence. If the string is empty, then a only a carriage-return and linefeed are written to the socket. Note that if the string contains a null character, any data that follows the null character will be discarded.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

The **WriteLine** method should only be used to send text, never binary data. In particular, this method will discard any data that follows a null character and will append linefeed and carriage return control characters to the data stream. Calling this method will force the current thread to block until the complete line of text has been written, the write operation times out or the remote host aborts the connection. If this method is called with the **Blocking** property set to **false**, it will automatically switch the socket into a blocking mode, send the data and then restore the socket to non-blocking mode. If another socket operation is attempted while the **WriteLine** method is blocked sending data to the remote host, an error will occur. It is recommended that this method only be used with blocking socket connections.

The **Write** and **WriteLine** methods can be safely intermixed.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.WriteLine Overload List

SocketWrench.WriteLine Method (String, Int32)

Send a line of text to the remote host, terminated by a carriage-return and linefeed.

[Visual Basic]

```
Overloads Public Function WriteLine( _
   ByVal buffer As String, _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool WriteLine(
   string buffer,
   ref int length
);
```

Parameters

buffer

A string which contains the data that will be sent to the remote host. The data will always be terminated with a carriage-return and linefeed control character sequence. If the string is empty, then a only a carriage-return and linefeed are written to the socket. Note that if the string contains a null character, any data that follows the null character will be discarded.

length

An integer value which specifies the maximum number of characters to write. This value cannot be larger than the length of the string specified by the caller.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

The **WriteLine** method should only be used to send text, never binary data. In particular, this method will discard any data that follows a null character and will append linefeed and carriage return control characters to the data stream. Calling this method will force the current thread to block until the complete line of text has been written, the write operation times out or the remote host aborts the connection. If this method is called with the **Blocking** property set to **false**, it will automatically switch the socket into a blocking mode, send the data and then restore the socket to non-blocking mode. If another socket operation is attempted while the **WriteLine** method is blocked sending data to the remote host, an error will occur. It is recommended that this method only be used with blocking socket connections.

The **Write** and **WriteLine** methods can be safely intermixed.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.WriteLine Overload List

SocketWrench.WriteStream Method

Write a stream of bytes to the socket.

Overload List

Write a stream of bytes to the socket.

public bool WriteStream(byte[],ref int);

Write a string of characters to the socket.

public bool WriteStream(string,ref int);

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.WriteStream Method (Byte[], Int32)

Write a stream of bytes to the socket.

[Visual Basic]

```
Overloads Public Function WriteStream( _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool WriteStream(
   byte[] buffer,
   ref int length
);
```

Parameters

buffer

A byte array that contains the data to be written to the socket.

length

An integer value passed by reference which specifies the maximum number of bytes to write. This value cannot be larger than the size of the buffer specified by the caller. When the method returns, this value will be updated with the actual number of bytes written to the socket.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

The **WriteStream** method enables an application to write an arbitrarily large stream of data from a byte array to the socket. Unlike the **Write** method, which may not write all of the data in a single call, the **WriteStream** method will only return when all of the data has been written or an error occurs.

This method will force the current thread to block until the operation completes. If this method is called with the **Blocking** property set to **false**, it will automatically switch the socket into a blocking mode, write the data stream and then restore the socket to non-blocking mode when it has finished. If another socket operation is attempted while **WriteStream** is blocked sending data to the remote host, an error will occur. It is recommended that this method only be used with blocking (synchronous) socket connections; if the application needs to establish multiple simultaneous connections, it should create worker threads to manage each connection.

It is possible that some data will have been written to the socket even if the method returns **false**. Applications should also check the value of the *length* argument to determine if any data was sent. For example, if a timeout occurs while the function is waiting to write more data, it will return zero; however, some data may have already been written to the socket prior to the error condition.

Because **WriteStream** can potentially cause the current thread to block for long periods of time as the data stream is being written, the class instance will periodically generate **OnProgress** events. An application can use this event to update the user interface as the data is being written. Note that an application should never perform a blocking operation inside the event handler.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.WriteStream Overload List

SocketWrench.WriteStream Method (String, Int32)

Write a string of characters to the socket.

[Visual Basic]

```
Overloads Public Function WriteStream( _
   ByVal buffer As String, _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool WriteStream(
   string buffer,
   ref int length
);
```

Parameters

buffer

A string that contains the data to be written to the socket.

length

An integer value passed by reference which specifies the maximum number of characters to write. This value cannot be larger than the length of the string specified by the caller. When the method returns, this value will be updated with the actual number of bytes written to the socket.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

The **WriteStream** method enables an application to write an arbitrarily large stream of data from a string to the socket. Unlike the **Write** method, which may not write all of the data in a single call, the **WriteStream** method will only return when all of the data has been written or an error occurs.

This method will force the current thread to block until the operation completes. If this method is called with the **Blocking** property set to **false**, it will automatically switch the socket into a blocking mode, write the data stream and then restore the socket to non-blocking mode when it has finished. If another socket operation is attempted while **WriteStream** is blocked sending data to the remote host, an error will occur. It is recommended that this method only be used with blocking (synchronous) socket connections; if the application needs to establish multiple simultaneous connections, it should create worker threads to manage each connection.

It is possible that some data will have been written to the socket even if the method returns **false**. Applications should also check the value of the *length* argument to determine if any data was sent. For example, if a timeout occurs while the function is waiting to write more data, it will return zero; however, some data may have already been written to the socket prior to the error condition.

Because **WriteStream** can potentially cause the current thread to block for long periods of time as the data stream is being written, the class instance will periodically generate **OnProgress** events. An application can use this event to update the user interface as the data is being written. Note that an application should never perform a blocking operation inside the event handler.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.WriteStream Overload List

SocketWrench.WriteTo Method

Write one or more bytes of data to the socket.

Overload List

Write one or more bytes of data to the socket.

public int WriteTo(byte[],int,string,int);

Write one or more bytes of data to the socket.

public int WriteTo(byte[],string,int);

Write a string of characters to the socket.

public int WriteTo(string,int,string,int);

Write a string of characters to the socket.

public int WriteTo(string,string,int);

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.WriteTo Method (Byte[], Int32, String, Int32)

Write one or more bytes of data to the socket.

[Visual Basic]

```
Overloads Public Function WriteTo( _
   ByVal buffer As Byte(), _
   ByVal length As Integer, _
   ByVal hostAddress As String, _
   ByVal hostPort As Integer _
) As Integer
```

[C#]

```
public int WriteTo(
   byte[] buffer,
   int length,
   string hostAddress,
   int hostPort
);
```

Parameters

buffer
A byte array that contains the data to be written to the socket.

length
An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the buffer specified by the caller.

hostAddress
A string value which specifies the address of the remote host that the data will be sent to. For datagram sockets, this may be any valid Internet address. For stream sockets, this must be the same address that was used to establish the connection.

hostPort
An integer value which specifies the port number of the remote host that the data will be sent to. For datagram sockets, this may be any valid port number. For stream sockets, this must be the same port number that was used to establish the connection.

Return Value

An integer value which specifies the number of bytes actually written to the socket. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **WriteTo** method sends one or more bytes of data to the remote host. If there is enough room in the socket's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the socket is in blocking mode, then the method will block until the data can be sent. If the socket is in non-blocking mode and the send buffer is full, an error will occur.

This method is typically used when writing binary data to a datagram socket.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.WriteTo Overload List

SocketWrench.WriteTo Method (Byte[], String, Int32)

Write one or more bytes of data to the socket.

[Visual Basic]

```
Overloads Public Function WriteTo( _
   ByVal buffer As Byte(), _
   ByVal hostAddress As String, _
   ByVal hostPort As Integer _
) As Integer
```

[C#]

```
public int WriteTo(
   byte[] buffer,
   string hostAddress,
   int hostPort
);
```

Parameters

buffer
A byte array that contains the data to be written to the socket.

hostAddress
A string value which specifies the address of the remote host that the data will be sent to. For datagram sockets, this may be any valid Internet address. For stream sockets, this must be the same address that was used to establish the connection.

hostPort
An integer value which specifies the port number of the remote host that the data will be sent to. For datagram sockets, this may be any valid port number. For stream sockets, this must be the same port number that was used to establish the connection.

Return Value

An integer value which specifies the number of bytes actually written to the socket. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **WriteTo** method sends one or more bytes of data to the remote host. If there is enough room in the socket's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the socket is in blocking mode, then the method will block until the data can be sent. If the socket is in non-blocking mode and the send buffer is full, an error will occur.

This method is typically used when writing binary data to a datagram socket.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.WriteTo Overload List

SocketWrench.WriteTo Method (String, Int32, String, Int32)

Write a string of characters to the socket.

[Visual Basic]

```
Overloads Public Function WriteTo( _
   ByVal buffer As String, _
   ByVal length As Integer, _
   ByVal hostAddress As String, _
   ByVal hostPort As Integer _
) As Integer
```

[C#]

```
public int WriteTo(
   string buffer,
   int length,
   string hostAddress,
   int hostPort
);
```

Parameters

buffer

A string that contains the data to be written to the socket.

length

An integer value which specifies the maximum number of characters to write. This value cannot be larger than the length of the string specified by the caller.

hostAddress

A string value which specifies the address of the remote host that the data will be sent to. For datagram sockets, this may be any valid Internet address. For stream sockets, this must be the same address that was used to establish the connection.

hostPort

An integer value which specifies the port number of the remote host that the data will be sent to. For datagram sockets, this may be any valid port number. For stream sockets, this must be the same port number that was used to establish the connection.

Return Value

An integer value which specifies the number of bytes actually written to the socket. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **WriteTo** method sends a string of characters to the remote host. If there is enough room in the socket's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the socket is in blocking mode, then the method will block until the data can be sent. If the socket is in non-blocking mode and the send buffer is full, an error will occur.

This method is typically used when writing text data to a datagram socket.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.WriteTo Overload List

SocketWrench.WriteTo Method (String, String, Int32)

Write a string of characters to the socket.

[Visual Basic]

```
Overloads Public Function WriteTo( _
   ByVal buffer As String, _
   ByVal hostAddress As String, _
   ByVal hostPort As Integer _
) As Integer
```

[C#]

```
public int WriteTo(
   string buffer,
   string hostAddress,
   int hostPort
);
```

Parameters

buffer
A string that contains the data to be written to the socket.

hostAddress
A string value which specifies the address of the remote host that the data will be sent to. For datagram sockets, this may be any valid Internet address. For stream sockets, this must be the same address that was used to establish the connection.

hostPort
An integer value which specifies the port number of the remote host that the data will be sent to. For datagram sockets, this may be any valid port number. For stream sockets, this must be the same port number that was used to establish the connection.

Return Value

An integer value which specifies the number of bytes actually written to the socket. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **WriteTo** method sends a string of characters to the remote host. If there is enough room in the socket's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the socket is in blocking mode, then the method will block until the data can be sent. If the socket is in non-blocking mode and the send buffer is full, an error will occur.

This method is typically used when writing text data to a datagram socket.

See Also

SocketWrench Class | SocketTools Namespace | SocketWrench.WriteTo Overload List

SocketWrench Events

The events of the **SocketWrench** class are listed below. For a complete list of **SocketWrench** class members, see the SocketWrench Members topic.

Public Instance Events

⚡ OnAccept	Occurs when a remote host attempts to establish a connection with the local system.
⚡ OnCancel	Occurs when a blocking socket operation is canceled.
⚡ OnConnect	Occurs when a connection is established with the remote host.
⚡ OnDisconnect	Occurs when the remote host disconnects from the local system.
⚡ OnError	Occurs when an socket operation fails.
⚡ OnProgress	Occurs as a data stream is being read or written to the socket.
⚡ OnRead	Occurs when data is available to be read from the socket.
⚡ OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
⚡ OnWrite	Occurs when data can be written to the socket.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.OnAccept Event

Occurs when a remote host attempts to establish a connection with the local system.

[Visual Basic]

```
Public Event OnAccept As OnAcceptEventHandler
```

[C#]

```
public event OnAcceptEventHandler OnAccept;
```

Event Data

The event handler receives an argument of type SocketWrench.AcceptEventArgs containing data related to this event. The following **SocketWrench.AcceptEventArgs** property provides information specific to this event.

Property	Description
Handle	Gets a value that specifies the socket handle for the listening server.

Remarks

The **OnAccept** event occurs when a remote host attempts to connect to the local system. A connection is not actually established until it has been accepted by the listening server. To accept the connection, the application must call the **Accept** method.

The **PeerAddress** or **PeerName** properties may be used to determine the Internet address and host name of the remote host that is establishing the connection. Note that this information may not be available until after the **Accept** method is called to accept the connection.

This event is only generated if the socket is in non-blocking mode.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.AcceptEventArgs Class

Provides data for the OnAccept event.

For a list of all members of this type, see SocketWrench.AcceptEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.SocketWrench.AcceptEventArgs

[Visual Basic]
```
Public Class SocketWrench.AcceptEventArgs
    Inherits EventArgs
```

[C#]
```
public class SocketWrench.AcceptEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

AcceptEventArgs specifies the socket handle for the server that should accept the incoming client connection.

The OnAccept event occurs when a remote host attempts to establish a connection with the local system.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SocketWrench (in SocketTools.SocketWrench.dll)

See Also

SocketWrench.AcceptEventArgs Members | SocketTools Namespace

SocketWrench.AcceptEventArgs Members

SocketWrench.AcceptEventArgs overview

Public Instance Constructors

SocketWrench.AcceptEventArgs Constructor	Initializes a new instance of the SocketWrench.AcceptEventArgs class.

Public Instance Properties

Handle	Gets a value that specifies the socket handle for the listening server.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

SocketWrench.AcceptEventArgs Class | SocketTools Namespace

SocketWrench.AcceptEventArgs Constructor

Initializes a new instance of the SocketWrench.AcceptEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public SocketWrench.AcceptEventArgs();
```

See Also

SocketWrench.AcceptEventArgs Class | SocketTools Namespace

SocketWrench.AcceptEventArgs Properties

The properties of the **SocketWrench.AcceptEventArgs** class are listed below. For a complete list of **SocketWrench.AcceptEventArgs** class members, see the SocketWrench.AcceptEventArgs Members topic.

Public Instance Properties

Handle	Gets a value that specifies the socket handle for the listening server.

See Also

SocketWrench.AcceptEventArgs Class | SocketTools Namespace

SocketWrench.AcceptEventArgs.Handle Property

Gets a value that specifies the socket handle for the listening server.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer value which specifies the server socket handle.

See Also

SocketWrench.AcceptEventArgs Class | SocketTools Namespace

SocketWrench.OnCancel Event

Occurs when a blocking socket operation is canceled.

[Visual Basic]

```
Public Event OnCancel As EventHandler
```

[C#]

```
public event EventHandler OnCancel;
```

Remarks

The **OnCancel** event is generated when a blocking socket operation, such as sending or receiving data, is canceled with the **Cancel** method. To assist in determining which operation was canceled, check the value of the **Status** property.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.OnConnect Event

Occurs when a connection is established with the remote host.

[Visual Basic]

```
Public Event OnConnect As EventHandler
```

[C#]

```
public event EventHandler OnConnect;
```

Remarks

The **OnConnect** event occurs when a connection is made with a remote host as a result of a **Connect** method call. When the **Connect** method is called and the **Blocking** property is set to **false**, a socket is created but the connection is not actually established until after this event occurs. Between the time connection process is started and this event fires, no operation may be performed on the socket other than calling the **Disconnect** method.

This event is only generated if the socket is in non-blocking mode.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.OnDisconnect Event

Occurs when the remote host disconnects from the local system.

[Visual Basic]

```
Public Event OnDisconnect As EventHandler
```

[C#]

```
public event EventHandler OnDisconnect;
```

Remarks

The **OnDisconnect** event occurs when the remote host closes its socket, terminating its connection with the application. Because there may still be data in the socket receive buffers, you should continue to read data from the socket until the **Read** method returns a value of 0. Once all of the data has been read, you should call the **Disconnect** method to close the local socket and terminate the session.

This event is only generated if the socket is in non-blocking mode.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.OnError Event

Occurs when an socket operation fails.

[Visual Basic]

```
Public Event OnError As OnErrorEventHandler
```

[C#]

```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type SocketWrench.ErrorEventArgs containing data related to this event. The following **SocketWrench.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a socket operation fails.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see SocketWrench.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.SocketWrench.ErrorEventArgs

[Visual Basic]
```
Public Class SocketWrench.ErrorEventArgs
    Inherits EventArgs
```

[C#]
```
public class SocketWrench.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SocketWrench (in SocketTools.SocketWrench.dll)

See Also

SocketWrench.ErrorEventArgs Members | SocketTools Namespace

SocketWrench.ErrorEventArgs Members

SocketWrench.ErrorEventArgs overview

Public Instance Constructors

SocketWrench.ErrorEventArgs Constructor	Initializes a new instance of the SocketWrench.ErrorEventArgs class.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

SocketWrench.ErrorEventArgs Class | SocketTools Namespace

SocketWrench.ErrorEventArgs Constructor

Initializes a new instance of the SocketWrench.ErrorEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public SocketWrench.ErrorEventArgs();
```

See Also

SocketWrench.ErrorEventArgs Class | SocketTools Namespace

SocketWrench.ErrorEventArgs Properties

The properties of the **SocketWrench.ErrorEventArgs** class are listed below. For a complete list of **SocketWrench.ErrorEventArgs** class members, see the SocketWrench.ErrorEventArgs Members topic.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

SocketWrench.ErrorEventArgs Class | SocketTools Namespace

SocketWrench.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

SocketWrench.ErrorEventArgs Class | SocketTools Namespace | Error Property

SocketWrench.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public SocketWrench.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

SocketWrench.ErrorEventArgs Class | SocketTools Namespace | Description Property

SocketWrench.OnProgress Event

Occurs as a data stream is being read or written to the socket.

[Visual Basic]

```
Public Event OnProgress As OnProgressEventHandler
```

[C#]

```
public event OnProgressEventHandler OnProgress;
```

Event Data

The event handler receives an argument of type SocketWrench.ProgressEventArgs containing data related to this event. The following **SocketWrench.ProgressEventArgs** properties provide information specific to this event.

Property	Description
BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
BytesTotal	Gets a value which specifies the total number of bytes in the data stream.
Percent	Gets a value which specifies the percentage of data that has been read or written.

Remarks

The **OnProgress** event occurs as a data stream is being read or written to the socket. If large amounts of data are being read or written, this event can be used to update a progress bar or other user-interface component to provide the user with some visual feedback on the progress of the operation.

This event is only generated when the **ReadStream**, **WriteStream** or **StoreStream** methods are called.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.ProgressEventArgs Class

Provides data for the OnProgress event.

For a list of all members of this type, see SocketWrench.ProgressEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.SocketWrench.ProgressEventArgs

[Visual Basic]
```
Public Class SocketWrench.ProgressEventArgs
    Inherits EventArgs
```

[C#]
```
public class SocketWrench.ProgressEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ProgressEventArgs specifies the number of bytes copied from the data stream, the total number of bytes in the data stream and a completion percentage.

The OnProgress event occurs as a data stream is being read or written to the socket. This event only occurs when the **ReadStream**, **WriteStream** or **StoreStream** methods are called.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SocketWrench (in SocketTools.SocketWrench.dll)

See Also

SocketWrench.ProgressEventArgs Members | SocketTools Namespace

SocketWrench.ProgressEventArgs Members

SocketWrench.ProgressEventArgs overview

Public Instance Constructors

SocketWrench.ProgressEventArgs Constructor	Initializes a new instance of the SocketWrench.ProgressEventArgs class.

Public Instance Properties

BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
BytesTotal	Gets a value which specifies the total number of bytes in the data stream.
Percent	Gets a value which specifies the percentage of data that has been read or written.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

SocketWrench.ProgressEventArgs Class | SocketTools Namespace

SocketWrench.ProgressEventArgs Constructor

Initializes a new instance of the SocketWrench.ProgressEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public SocketWrench.ProgressEventArgs();
```

See Also

SocketWrench.ProgressEventArgs Class | SocketTools Namespace

SocketWrench.ProgressEventArgs Properties

The properties of the **SocketWrench.ProgressEventArgs** class are listed below. For a complete list of **SocketWrench.ProgressEventArgs** class members, see the SocketWrench.ProgressEventArgs Members topic.

Public Instance Properties

BytesCopied	Gets a value which specifies the number of bytes of data that has been read or written.
BytesTotal	Gets a value which specifies the total number of bytes in the data stream.
Percent	Gets a value which specifies the percentage of data that has been read or written.

See Also

SocketWrench.ProgressEventArgs Class | SocketTools Namespace

SocketWrench.ProgressEventArgs.BytesCopied Property

Gets a value which specifies the number of bytes of data that has been read or written.

[Visual Basic]

```
Public ReadOnly Property BytesCopied As Integer
```

[C#]

```
public int BytesCopied {get;}
```

Property Value

An integer value which specifies the number of bytes of data.

Remarks

The **BytesCopied** property specifies the number of bytes that have been read from the socket and stored in the local stream buffer, or written from the stream buffer to the socket.

See Also

SocketWrench.ProgressEventArgs Class | SocketTools Namespace | BytesTotal Property | Percent Property

SocketWrench.ProgressEventArgs.BytesTotal Property

Gets a value which specifies the total number of bytes in the data stream.

[Visual Basic]

```
Public ReadOnly Property BytesTotal As Integer
```

[C#]

```
public int BytesTotal {get;}
```

Property Value

An integer value which specifies the number of bytes of data.

Remarks

The **BytesTotal** property specifies the total amount of data being read from the socket and stored in the data stream, or written from the data stream to the socket. If the amount of data was unknown or unspecified at the time the method call was made, then this value will always be the same as the **BytesCopied** property.

See Also

SocketWrench.ProgressEventArgs Class | SocketTools Namespace | BytesCopied Property | Percent Property

SocketWrench.ProgressEventArgs.Percent Property

Gets a value which specifies the percentage of data that has been read or written.

[Visual Basic]

```
Public ReadOnly Property Percent As Integer
```

[C#]

```
public int Percent {get;}
```

Property Value

An integer value which specifies a percentage.

Remarks

The **Percent** property specifies the percentage of data that has been transmitted, expressed as an integer value between 0 and 100, inclusive. If the maximum size of the data stream was not specified by the caller, this value will always be 100.

See Also

SocketWrench.ProgressEventArgs Class | SocketTools Namespace | BytesCopied Property | BytesTotal Property

SocketWrench.OnRead Event

Occurs when data is available to be read from the socket.

[Visual Basic]
```
Public Event OnRead As EventHandler
```

[C#]
```
public event EventHandler OnRead;
```

Remarks

The **OnRead** event occurs when data is available to be read from the socket. This event is level-triggered, which means that once this event fires, it will not occur again until some data has been read from the socket. This design prevents an application from being flooded with event notifications. It is recommended that your application read all of the available data from the socket and store it in a local buffer for processing. See the example below.

This event is only generated if the socket is in non-blocking mode.

Example

```
Private Sub Socket_OnRead(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Socket.OnRead
    Dim strBuffer As String
    Dim nRead As Integer

    Do
        ' Read up to m_nBufferSize bytes of data from the socket
        nRead = Socket.Read(strBuffer, m_nBufferSize)

        If nRead > 0 Then
            ' Append the data to an internal buffer for processing
            m_dataBuffer = m_dataBuffer + strBuffer
        End If
    Loop Until nRead < 1

    ProcessData()
End Sub
```

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.OnTimeout Event

Occurs when a blocking operation fails to complete before the timeout period elapses.

[Visual Basic]

```
Public Event OnTimeout As EventHandler
```

[C#]

```
public event EventHandler OnTimeout;
```

Remarks

The **OnTimeout** event occurs when a blocking operation, such as sending or receiving data on the socket, fails to complete before the specified timeout period elapses. The timeout period for a blocking operation can be adjusted by setting the **Timeout** property.

This event is only generated if the socket is in blocking mode.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.OnWrite Event

Occurs when data can be written to the socket.

[Visual Basic]

```
Public Event OnWrite As EventHandler
```

[C#]

```
public event EventHandler OnWrite;
```

Remarks

The **OnWrite** event occurs when the application can write data to the socket. This event will typically occur when a connection is first established with the remote host, and after the **Write** method has failed because there was insufficient memory available in the socket send buffers. In the second case, when some of the buffered data has been successfully sent to the remote host and there is space available in the send buffers, this event is used to signal the application that it may attempt to send more data.

This event is only generated if the socket is in non-blocking mode.

See Also

SocketWrench Class | SocketTools Namespace

SocketWrench.ErrorCode Enumeration

Specifies the error codes returned by the SocketWrench class.

[Visual Basic]

```
Public Enum SocketWrench.ErrorCode
```

[C#]

```
public enum SocketWrench.ErrorCode
```

Remarks

The SocketWrench class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidBuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SocketWrench (in SocketTools.SocketWrench.dll)

See Also

SocketTools Namespace

SocketWrench.SecureKeyAlgorithm Enumeration

Specifies the key exchange algorithms that the SocketWrench class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum SocketWrench.SecureKeyAlgorithm
```

[C#]
```
[Flags]
public enum SocketWrench.SecureKeyAlgorithm
```

Remarks

The SocketWrench class uses the **SecureKeyAlgorithm** enumeration to identify the key exchange algorithm that was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
keyExchangeNone	No key exchange algorithm has been selected. This is not a secure connection with the server.	0
keyExchangeRSA	The RSA public key exchange algorithm has been selected.	1
keyExchangeKEA	The KEA public key exchange algorithm has been selected. This is an improved version of the Diffie-Hellman public key algorithm.	2
keyExchangeDH	The Diffie-Hellman public key exchange algorithm has been selected.	4
keyExchangeECDH	The Elliptic Curve Diffie-Hellman key exchange algorithm was selected. This is a variant of the Diffie-Hellman algorithm which uses elliptic curve cryptography. This key exchange algorithm is only supported on Windows XP SP3 SP3 and later versions of the operating system.	8

Requirements

Namespace: SocketTools

Assembly: SocketTools.SocketWrench (in SocketTools.SocketWrench.dll)

See Also

SocketTools Namespace

SocketWrench.SecureCipherAlgorithm Enumeration

Specifies the encryption algorithms that the SocketWrench class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum SocketWrench.SecureCipherAlgorithm
```

[C#]
```
[Flags]
public enum SocketWrench.SecureCipherAlgorithm
```

Remarks

The SocketWrench class uses the **SecureCipherAlgorithm** enumeration to identify which encryption algorithm was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
cipherNone	No cipher has been selected. A secure connection has not been established with the remote host.	0
cipherRC2	The RC2 block cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.	1
cipherRC4	The RC4 stream cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.	2
cipherRC5	The RC5 block cipher was selected. This is a variable key length cipher which supports keys up to 2040 bits, in 8-bit increments.	4
cipherDES	The DES (Data Encryption Standard) block cipher was selected. This is a fixed key length cipher using 56-bit keys.	8
cipherDES3	The Triple DES block cipher was selected. This cipher encrypts the data three times using different keys, effectively using a 168-bit key length.	16
cipherDESX	A variant of the DES block cipher which XORs an extra 64-bits of the key before and after the plaintext has been encrypted, increasing the key size to 184 bits.	32
cipherAES	The Advanced Encryption Standard	64

	cipher (also known as the Rijndael cipher) is a fixed block size cipher which use a key size of 128, 192 or 256 bits. This cipher is supported on Windows XP SP3 SP3 and later versions of the operating system.	
cipherSkipjack	The Skipjack block cipher was selected. This is a fixed key length cipher, using 80-bit keys.	128

Requirements

Namespace: SocketTools

Assembly: SocketTools.SocketWrench (in SocketTools.SocketWrench.dll)

See Also

SocketTools Namespace

SocketWrench.SecureHashAlgorithm Enumeration

Specifies the hash algorithms that the SocketWrench class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum SocketWrench.SecureHashAlgorithm
```

```
[C#]
[Flags]
public enum SocketWrench.SecureHashAlgorithm
```

Remarks

The SocketWrench class uses the **SecureHashAlgorithm** enumeration to identify the message digest (hash) algorithm that was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
hashNone	No hash algorithm has been selected. This is not a secure connection with the server.	0
hashMD5	The MD5 algorithm was selected. This algorithm produces a 128-bit message digest. This algorithm is no longer considered to be cryptographically secure.	1
hashSHA	The SHA-1 algorithm was selected. This algorithm produces a 160-bit message digest. This algorithm is no longer considered to be cryptographically secure.	2
hashSHA256	The SHA-256 algorithm was selected. This algorithm produces a 256-bit message digest.	4
hashSHA384	The SHA-384 algorithm was selected. This algorithm produces a 384-bit message digest.	8
hashSHA512	The SHA-512 algorithm was selected. This algorithm produces a 512-bit message digest.	16

Requirements

Namespace: SocketTools

Assembly: SocketTools.SocketWrench (in SocketTools.SocketWrench.dll)

See Also

SocketTools Namespace

SocketWrench.SecurityCertificate Enumeration

Specifies the security certificate status values that may be returned by the SocketWrench class.

[Visual Basic]

```
Public Enum SocketWrench.SecurityCertificate
```

[C#]

```
public enum SocketWrench.SecurityCertificate
```

Remarks

The SocketWrench class uses the **SecurityCertificate** enumeration to identify the current status of the certificate that was provided by the remote host when a secure connection was established.

Members

Member Name	Description
certificateNone	No certificate information is available. A secure connection was not established with the server.
certificateValid	The certificate is valid.
certificateNoMatch	The certificate is valid, however the domain name specified in the certificate does not match the domain name of the remote host. The application can examine the **CertificateSubject** property to determine the site the certificate was issued to.
certificateExpired	The certificate has expired and is no longer valid. The application can examine the **CertificateExpires** property to determine when the certificate expired.
certificateRevoked	The certificate has been revoked and is no longer valid. It is recommended that the application immediately terminate the connection if this status is returned.
certificateUntrusted	The certificate has not been issued by a trusted authority, or the certificate is not trusted on the local host. It is recommended that the application immediately terminate the connection if this status is returned.
certificateInvalid	The certificate is invalid. This typically indicates that the internal structure of the certificate is damaged. It is recommended that the application immediately terminate the connection if this status is returned.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SocketWrench (in SocketTools.SocketWrench.dll)

See Also

SocketTools Namespace

SocketWrench.SecurityProtocols Enumeration

Specifies the security protocols that the SocketWrench class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum SocketWrench.SecurityProtocols
```

[C#]
```
[Flags]
public enum SocketWrench.SecurityProtocols
```

Remarks

The SocketWrench class uses the **SecurityProtocols** enumeration to specify one or more security protocols to be used when establishing a connection with a remote host. Multiple protocols may be specified if necessary and the actual protocol used will be negotiated with the remote host. It is recommended that most applications use **protocolDefault** when creating a secure connection.

Members

Member Name	Description	Value
protocolNone	No security protocol will be used, a secure connection will not be established.	0
protocolSSL2	The SSL 2.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections.	1
protocolSSL3	The SSL 3.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections. In most cases, this protocol is only selected if TLS is not supported by the server.	2
protocolTLS10	The TLS 1.0 protocol has been selected. This version of the protocol is commonly used by older servers and is the only version of TLS supported on Windows platforms prior to Windows 7 SP1 and Windows Server 2008 R2.	4
protocolTLS11	The TLS 1.1 protocol has been selected. This version of TLS is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of the	8

	operating system.	
protocolTLS12	The TLS 1.2 protocol has been selected. This is the default version of the protocol and is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of Windows. It is recommended that you use this version of TLS.	16
protocolSSL	Any version of the Secure Sockets Layer (SSL) protocol should be used. The actual protocol version used will be negotiated with the remote host.	3
protocolTLS	Any version of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with with preference for TLS 1.2.	28
protocolTLS1	Version 1.0, 1.1 or 1.2 of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with preference for TLS 1.2.	28
protocolDefault	The default selection of security protocols will be used when establishing a connection. The TLS 1.2, 1.1 and 1.0 protocols will be negotiated with the host, in that order of preference. This option will always request the latest version of the preferred security protocols and is the recommended value.	16
protocolUnknown	An unknown or unsupported security protocol has been specified. This value indicates an error condition.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.SocketWrench (in SocketTools.SocketWrench.dll)

See Also

SocketTools Namespace

SocketWrench.ShutdownOptions Enumeration

Specifies the shutdown options that the SocketWrench class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum SocketWrench.ShutdownOptions
```

[C#]

```
[Flags]
public enum SocketWrench.ShutdownOptions
```

Remarks

The SocketWrench class uses the **ShutdownOptions** enumeration to specify how reading and writing on the socket should be handled when the **Shutdown** method is called.

Members

Member Name	Description	Value
shutdownRead	Disable any further reading of data. The application will be able to continue to send data. The remote host will see this as the connection being closed.	0
shutdownWrite	Disable any further sending of data. The application will be able to continue to read data until the remote host closes the connection.	1
shutdownReadWrite	Disable any further reading or writing to the socket. The remote host will see this as the connection being closed.	2

Requirements

Namespace: SocketTools

Assembly: SocketTools.SocketWrench (in SocketTools.SocketWrench.dll)

See Also

SocketTools Namespace

SocketWrench.SocketByteOrder Enumeration

Specifies the byte-order in which integer data may exchanged with a remote host.

[Visual Basic]

```
Public Enum SocketWrench.SocketByteOrder
```

[C#]

```
public enum SocketWrench.SocketByteOrder
```

Remarks

The byte-order is used to specify how 16-bit (short) integer and 32-bit (long) integer data is written to and read from the socket.

Members

Member Name	Description
byteOrderNative	Integer data will be sent and received using the native byte order.
byteOrderNetwork	Integer data will be sent and received using network byte order.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SocketWrench (in SocketTools.SocketWrench.dll)

See Also

SocketTools Namespace

SocketWrench.SocketOptions Enumeration

Specifies the options that the SocketWrench class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum SocketWrench.SocketOptions
```

[C#]
```
[Flags]
public enum SocketWrench.SocketOptions
```

Remarks

The SocketWrench class uses the **SocketOptions** enumeration to specify one or more options to be used when establishing a connection with a remote host. Multiple options may be specified if necessary.

Members

Member Name	Description	Value
optionNone	No option specified.	0
optionBroadcast	This option specifies that broadcasting should be enabled for datagrams. This option is invalid for stream sockets.	1
optionDontRoute	This option specifies default routing should not be used. This option should not be specified unless absolutely necessary.	2
optionKeepAlive	This option specifies that packets are to be sent to the remote system when no data is being exchanged to keep the connection active. This option is only valid for stream sockets.	4
optionReuseAddress	This option specifies the local address can be reused. This option is commonly used by server applications.	8
optionNoDelay	This option disables the Nagle algorithm, which buffers unacknowledged data and insures that a full-size packet can be sent to the remote host.	16
optionInLine	This option specifies that out-of-band data should be received inline with the standard data stream. This option is only valid for stream sockets.	32
optionTrustedSite	This option specifies the sever should be trusted. The server certificate will not be validated and the connection will always	2048

	be permitted. This option only affects secure client connections.	
optionSecure	This option specifies that a secure, encrypted connection will be established with the remote host.	4096
optionSecureFallback	This option specifies the class should permit the use of less secure cipher suites for compatibility with legacy clients and servers. If this option is specified, it will enable connections using TLS 1.0 and cipher suites that use RC4, MD5 and SHA1.	32768
optionPreferIPv6	This option specifies the client should prefer the use of IPv6 if the remote hostname can be resolved to both an IPv6 and IPv4 address. This option is ignored if the local system does not have IPv6 enabled, or when the hostname can only be resolved to an IPv4 address. If the server hostname can only be resolved to an IPv6 address, the client will attempt to establish a connection using IPv6 regardless if this option has been specified.	262144
optionFreeThread	This option specifies that class methods may be called from any thread, and not only the thread that established the connection. Using this option disables certain internal safety checks that are made by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads.	524288

Requirements

Namespace: SocketTools

Assembly: SocketTools.SocketWrench (in SocketTools.SocketWrench.dll)

See Also

SocketTools Namespace

SocketWrench.SocketProtocol Enumeration

Specifies the protocols that the SocketWrench class supports.

[Visual Basic]
```
Public Enum SocketWrench.SocketProtocol
```

[C#]
```
public enum SocketWrench.SocketProtocol
```

Remarks

The SocketWrench class uses the **SocketProtocol** enumeration to specify which network protocol will be used when a socket is created. The default protocol used by the class is **socketStream**.

Members

Member Name	Description
socketStream	Transmission Control Protocol (TCP). This protocol should be used with stream sockets, where data is sent and received as an arbitrary stream of bytes.
socketDatagram	User Datagram Protocol (UDP). This protocol should be used with datagram sockets, where data is sent and received in discrete packets.
socketRaw	Raw sockets. This socket type is for special purpose applications which need access to the IP datagram.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SocketWrench (in SocketTools.SocketWrench.dll)

See Also

SocketTools Namespace

SocketWrench.SocketStatus Enumeration

Specifies the status values that may be returned by the SocketWrench class.

[Visual Basic]
```
Public Enum SocketWrench.SocketStatus
```

[C#]
```
public enum SocketWrench.SocketStatus
```

Remarks

The SocketWrench class uses the **SocketStatus** enumeration to identify the current status of the socket.

Members

Member Name	Description
statusUnused	A socket has not been created. Attempts to perform any network operations, such as sending or receiving data, will generate an error.
statusIdle	A socket has been created, but is not currently in use. A blocking socket operation can be executed at this point.
statusListen	The socket is listening for connections from remote hosts.
statusConnect	The socket is in the process of establishing a connection with a remote host.
statusAccept	The socket is in the process of accepting a connection from a remote client.
statusRead	The socket is in the process of receiving data from a remote host.
statusWrite	The socket is in the process of sending data to a remote host.
statusFlush	The control buffers are in the process of being flushed. Any data in the socket receive buffers will be discarded.
statusDisconnect	The socket is being closed and subsequent attempts to access the socket will result in an error.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SocketWrench (in SocketTools.SocketWrench.dll)

See Also

SocketTools Namespace

SocketWrench.SocketStream Enumeration

Specifies the data stream options that the SocketWrench class supports.

[Visual Basic]

```
Public Enum SocketWrench.SocketStream
```

[C#]

```
public enum SocketWrench.SocketStream
```

Remarks

The SocketWrench class uses the **SocketStream** enumeration to specify how data should be processed when read from a socket using either the **ReadStream** or **StoreStream** methods.

Members

Member Name	Description
streamDefault	The data stream will be returned to the caller unmodified. This option should always be used with binary data or data being stored in a byte array. If no options are specified, this is the default option used by this method.
streamConvert	The data stream is considered to be textual and will be modified so that end-of-line character sequences are converted to follow standard Windows conventions. This will ensure that all lines of text are terminated with a carriage-return and linefeed sequence. Because this option modifies the data stream, it should never be used with binary data. Using this option may result in the amount of data returned in the buffer to be larger than the source data. For example, if the source data only terminates a line of text with a single linefeed, this option will have the effect of inserting a carriage-return character before each linefeed.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SocketWrench (in SocketTools.SocketWrench.dll)

See Also

SocketTools Namespace | ReadStream Method | StoreStream Method

SocketWrench.TraceOptions Enumeration

Specifies the logging options that the SocketWrench class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum SocketWrench.TraceOptions
```

[C#]
```
[Flags]
public enum SocketWrench.TraceOptions
```

Remarks

The SocketWrench class uses the **TraceOptions** enumeration to specify what kind of debugging information is written to the trace logfile. These options are only meaningful when trace logging is enabled by setting the **Trace** property to **true**.

Members

Member Name	Description	Value
traceDefault	The default trace logging option. This is the same as specifying the **traceInfo** option.	0
traceInfo	All network function calls are written to the trace file. This is the default value.	0
traceError	Only those network function calls which fail are recorded in the trace file.	1
traceWarning	Only those network function calls which fail, or return values which indicate a warning, are recorded in the trace file.	2
traceHexDump	All network function calls are written to the trace file, plus all the data that is sent or received is displayed, in both ASCII and hexadecimal format.	4
traceProcess	All function calls in the current process are logged, rather than only those functions in the current thread. This option is useful for multithreaded applications that are using worker threads.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.SocketWrench (in SocketTools.SocketWrench.dll)

See Also

SocketTools Namespace

SocketWrench.OnAcceptEventHandler Delegate

Represents the method that will handle the OnAccept event.

[Visual Basic]

```
Public Delegate Sub SocketWrench.OnAcceptEventHandler( _
   ByVal sender As Object, _
   ByVal e As AcceptEventArgs _
)
```

[C#]

```
public delegate void SocketWrench.OnAcceptEventHandler(
      object sender,
      AcceptEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An AcceptEventArgs that contains the event data.

Remarks

When you create an **OnAcceptEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnAcceptEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SocketWrench (in SocketTools.SocketWrench.dll)

See Also

SocketTools Namespace | Accept Method | OnAccept Event

SocketWrench.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub SocketWrench.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void SocketWrench.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ErrorEventArgs that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SocketWrench (in SocketTools.SocketWrench.dll)

See Also

SocketTools Namespace

SocketWrench.OnProgressEventHandler Delegate

Represents the method that will handle the OnProgress event.

[Visual Basic]

```
Public Delegate Sub SocketWrench.OnProgressEventHandler( _
   ByVal sender As Object, _
   ByVal e As ProgressEventArgs _
)
```

[C#]

```
public delegate void SocketWrench.OnProgressEventHandler(
      object sender,
      ProgressEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A ProgressEventArgs that contains the event data.

Remarks

When you create an **OnProgressEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnProgressEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SocketWrench (in SocketTools.SocketWrench.dll)

See Also

SocketTools Namespace

SocketWrench.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see SocketWrench.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.SocketWrench.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class SocketWrench.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class SocketWrench.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketWrench can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the SocketWrench class.

Example

```
<Assembly: SocketTools.SocketWrench.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.SocketWrench.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.SocketWrench (in SocketTools.SocketWrench.dll)

See Also

SocketWrench.RuntimeLicenseAttribute Members | SocketTools Namespace

SocketWrench.RuntimeLicenseAttribute Members

SocketWrench.RuntimeLicenseAttribute overview

Public Instance Constructors

SocketWrench.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

SocketWrench.RuntimeLicenseAttribute Class | SocketTools Namespace

SocketWrench.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public SocketWrench.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketWrench can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the SocketWrench class.

See Also

SocketWrench.RuntimeLicenseAttribute Class | SocketTools Namespace

SocketWrench.RuntimeLicenseAttribute Properties

The properties of the **SocketWrench.RuntimeLicenseAttribute** class are listed below. For a complete list of **SocketWrench.RuntimeLicenseAttribute** class members, see the SocketWrench.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

SocketWrench.RuntimeLicenseAttribute Class | SocketTools Namespace

SocketWrench.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

SocketWrench.RuntimeLicenseAttribute Class | SocketTools Namespace

SocketWrenchException Class

The exception that is thrown when a socket error occurs.

For a list of all members of this type, see SocketWrenchException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.SocketWrenchException

[Visual Basic]

```
Public Class SocketWrenchException
    Inherits ApplicationException
```

[C#]

```
public class SocketWrenchException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A SocketWrenchException is thrown by the SocketWrench class when an error occurs.

The default constructor for the SocketWrenchException class sets the **ErrorCode** property to the last socket error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SocketWrench (in SocketTools.SocketWrench.dll)

See Also

SocketWrenchException Members | SocketTools Namespace

SocketWrenchException Members

SocketWrenchException overview

Public Instance Constructors

SocketWrenchException	Overloaded. Initializes a new instance of the SocketWrenchException class.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

SocketWrenchException Class | SocketTools Namespace

SocketWrenchException Constructor

Initializes a new instance of the SocketWrenchException class with the last network error code.

Overload List

Initializes a new instance of the SocketWrenchException class with the last network error code.

public SocketWrenchException();

Initializes a new instance of the SocketWrenchException class with a specified error number.

public SocketWrenchException(int);

Initializes a new instance of the SocketWrenchException class with a specified error message.

public SocketWrenchException(string);

Initializes a new instance of the SocketWrenchException class with a specified error message and a reference to the inner exception that is the cause of this exception.

public SocketWrenchException(string,Exception);

See Also

SocketWrenchException Class | SocketTools Namespace

SocketWrenchException Constructor ()

Initializes a new instance of the SocketWrenchException class with the last network error code.

[Visual Basic]
```
Overloads Public Sub New()
```

[C#]
```
public SocketWrenchException();
```

Remarks

The ctor constructor sets the **ErrorCode** property to the last socket error that occurred. For more information about the errors that may occur, refer to the SocketWrench.ErrorCode enumeration.

See Also

SocketWrenchException Class | SocketTools Namespace | SocketWrenchException Constructor Overload List

SocketWrenchException Constructor (String)

Initializes a new instance of the SocketWrenchException class with a specified error message.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]

```
public SocketWrenchException(
   string message
);
```

Parameters

message

The error message that explains the reason for the exception.

Remarks

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

SocketWrenchException Class | SocketTools Namespace | SocketWrenchException Constructor Overload List

SocketWrenchException Constructor (String, Exception)

Initializes a new instance of the SocketWrenchException class with a specified error message and a reference to the inner exception that is the cause of this exception.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal innerException As Exception _
)
```

[C#]

```
public SocketWrenchException(
   string message,
   Exception innerException
);
```

Parameters

message

The error message that explains the reason for the exception.

innerException

The exception that is the cause of the current exception. If the ***innerException*** parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

SocketWrenchException Class | SocketTools Namespace | SocketWrenchException Constructor Overload List

SocketWrenchException Constructor (Int32)

Initializes a new instance of the SocketWrenchException class with a specified error number.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

[C#]

```
public SocketWrenchException(
   int code
);
```

Parameters

code

An integer value which specifies an error code.

Remarks

This constructor sets the **ErrorCode** property to the specified error code. For more information about the errors that may occur, refer to the SocketWrench.ErrorCode enumeration.

See Also

SocketWrenchException Class | SocketTools Namespace | SocketWrenchException Constructor Overload List

SocketWrenchException Properties

The properties of the **SocketWrenchException** class are listed below. For a complete list of **SocketWrenchException** class members, see the SocketWrenchException Members topic.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

SocketWrenchException Class | SocketTools Namespace

SocketWrenchException.ErrorCode Property

Gets a value which specifies the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property ErrorCode As ErrorCode
```

[C#]

```
public SocketWrench.ErrorCode ErrorCode {get;}
```

Property Value

Returns a SocketWrench.ErrorCode enumeration value which specifies the error code.

Remarks

The **ErrorCode** property returns the error code that specifies the cause of the exception.

The default constructor for the SocketWrenchException class sets the error code to the last network error that occurred. For more information about the errors that may occur, refer to the SocketWrench.ErrorCode enumeration.

See Also

SocketWrenchException Class | SocketTools Namespace

SocketWrenchException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]

```
Overrides Public ReadOnly Property Message As String
```

[C#]

```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

SocketWrenchException Class | SocketTools Namespace

SocketWrenchException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception. This value is the same as the values returned by the Windows Sockets API. For more information about socket error codes, see the Windows Socket Version 2 API error code documentation in MSDN.

See Also

SocketWrenchException Class | SocketTools Namespace

SocketWrenchException Methods

The methods of the **SocketWrenchException** class are listed below. For a complete list of **SocketWrenchException** class members, see the SocketWrenchException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

SocketWrenchException Class | SocketTools Namespace

SocketWrenchException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]
```
Overrides Public Function ToString() As String
```

[C#]
```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

SocketWrenchException Class | SocketTools Namespace

SshClient Class

Implements the Secure Shell protocol.

For a list of all members of this type, see SshClient Members.

System.Object
  SocketTools.SshClient

[Visual Basic]
```
Public Class SshClient
    Implements IDisposable
```

[C#]
```
public class SshClient : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The Secure Shell (SSH) protocol is used to establish a secure connection with a server which provides a virtual terminal session for a user. Its functionality is similar to how character based consoles and serial terminals work, enabling a user to login to the server, execute commands and interact with applications running on the remote host. The class provides an interface for establishing the connection and handling the standard I/O functions needed by the program.

This class also includes methods which enable a program to easily execute a command on the server and scan the output for specific sequences of characters, making it very simple to write light-weight client interfaces to remote applications. This class can be combined with the **SocketTools.Terminal** component to provide complete terminal emulation services for a standard ANSI or DEC-VT220 terminal.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SshClient (in SocketTools.SshClient.dll)

See Also

SshClient Members | SocketTools Namespace

SshClient Members

SshClient overview

Public Static (Shared) Fields

sshPortDefault	A constant value which specifies the default port number.
sshTimeout	A constant value which specifies the default timeout period.

Public Instance Constructors

SshClient Constructor	Initializes a new instance of the SshClient class.

Public Instance Properties

AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
Blocking	Gets and sets a value which indicates if the client is in blocking mode.
CipherStrength	Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.
CodePage	Gets and sets the code page used when reading and writing text.
Columns	Gets and sets the number of columns for the virtual terminal session.
Command	Gets and sets the command that will be executed on the server.
ExitCode	Returns the exit code from the command executed on the server.
Fingerprint	Returns a string that uniquely identifies the server.
Handle	Gets a value that specifies the client handle allocated for the current session.
HashStrength	Gets a value which specifies the length of the message digest that was selected for a secure connection.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsConnected	Gets a value which indicates if a connection to the remote host has been established.

IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsReadable	Gets a value which indicates if there is data available to be read from the socket connection to the server.
IsWritable	Gets a value which indicates if data can be written to the client without blocking.
KeepAlive	Gets and sets a value which determines if the client session should kept active.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets the local Internet address that the client is bound to.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
NewLine	Gets and sets the end-of-line character sequences sent to the server.
Options	Gets and sets a value which specifies one or more client options.
Password	Gets and sets the password used to authenticate the client session.
PrivateKey	Gets and sets the name of the private key file used to authenticate the client session.
ProxyHost	Gets and sets the hostname or IP address of a proxy server.
ProxyPassword	Gets and sets the password used to authenticate the connection to a proxy server.
ProxyPort	Gets and sets a value that specifies the proxy server port number.
ProxyType	Gets and sets the type of proxy server the client will use to establish a connection.
ProxyUser	Gets and sets the username used to authenticate the connection to a proxy server.
RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.
Rows	Gets and sets the number of rows for the virtual terminal session.

Secure	Gets and sets a value which specifies if a secure connection is established.
SecureCipher	Gets a value that specifies the encryption algorithm used for a secure connection.
SecureHash	Gets a value that specifies the message digest algorithm used for a secure connection.
SecureKeyExchange	Gets a value that specifies the key exchange algorithm used for a secure connection.
SecureProtocol	Gets and sets a value which specifies the protocol used for a secure connection.
Status	Gets a value which specifies the current status of the client.
Terminal	Gets and sets the terminal type used for a remote login session.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
UserName	Gets and sets the username used to authenticate the client session.
Version	Gets a value which returns the current version of the SshClient class library.

Public Instance Methods

AttachThread	Attach an instance of the class to the current thread
Break	Sends a break signal to the remote host.
Cancel	Cancel the current blocking client operation.
Connect	Overloaded. Establish a connection with a remote host.
Control	Send a control message to the server.
Disconnect	Terminate the connection with a remote host.
Dispose	Overloaded. Releases all resources used by

	SshClient.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
Execute	Overloaded. Execute a command on the server and return the output.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the SshClient class.
Peek	Overloaded. Read data returned by the server, but do not remove it from the receive buffer.
Read	Overloaded. Read data from the server and store it in a byte array.
ReadLine	Overloaded. Read up to a line of data from the server and return it in a string buffer.
Reset	Reset the internal state of the object, resetting all properties to their default values.
Search	Overloaded. Search for a specific character sequence in the data stream.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
Write	Overloaded. Write one or more bytes of data to the server.
WriteLine	Overloaded. Send a line of text to the server, terminated by a carriage-return and linefeed.

Public Instance Events

OnCancel	Occurs when a blocking client operation is canceled.
OnConnect	Occurs when a connection is established with the remote host.
OnDisconnect	Occurs when the remote host disconnects from the local system.
OnError	Occurs when an client operation fails.
OnRead	Occurs when data is available to be read from the client.
OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
OnWrite	Occurs when data can be written to the client.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the SshClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

SshClient Class | SocketTools Namespace

SshClient Constructor

Initializes a new instance of the SshClient class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public SshClient();
```

See Also

SshClient Class | SocketTools Namespace

SshClient Properties

The properties of the **SshClient** class are listed below. For a complete list of **SshClient** class members, see the SshClient Members topic.

Public Instance Properties

AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
Blocking	Gets and sets a value which indicates if the client is in blocking mode.
CipherStrength	Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.
CodePage	Gets and sets the code page used when reading and writing text.
Columns	Gets and sets the number of columns for the virtual terminal session.
Command	Gets and sets the command that will be executed on the server.
ExitCode	Returns the exit code from the command executed on the server.
Fingerprint	Returns a string that uniquely identifies the server.
Handle	Gets a value that specifies the client handle allocated for the current session.
HashStrength	Gets a value which specifies the length of the message digest that was selected for a secure connection.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsConnected	Gets a value which indicates if a connection to the remote host has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsReadable	Gets a value which indicates if there is data available to be read from the socket connection to the server.
IsWritable	Gets a value which indicates if data can be written to the client without blocking.
KeepAlive	Gets and sets a value which determines if the client

	session should kept active.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets the local Internet address that the client is bound to.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
NewLine	Gets and sets the end-of-line character sequences sent to the server.
Options	Gets and sets a value which specifies one or more client options.
Password	Gets and sets the password used to authenticate the client session.
PrivateKey	Gets and sets the name of the private key file used to authenticate the client session.
ProxyHost	Gets and sets the hostname or IP address of a proxy server.
ProxyPassword	Gets and sets the password used to authenticate the connection to a proxy server.
ProxyPort	Gets and sets a value that specifies the proxy server port number.
ProxyType	Gets and sets the type of proxy server the client will use to establish a connection.
ProxyUser	Gets and sets the username used to authenticate the connection to a proxy server.
RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.
Rows	Gets and sets the number of rows for the virtual terminal session.
Secure	Gets and sets a value which specifies if a secure connection is established.
SecureCipher	Gets a value that specifies the encryption algorithm used for a secure connection.
SecureHash	Gets a value that specifies the message digest algorithm used for a secure connection.
SecureKeyExchange	Gets a value that specifies the key exchange algorithm used for a secure connection.

SecureProtocol	Gets and sets a value which specifies the protocol used for a secure connection.
Status	Gets a value which specifies the current status of the client.
Terminal	Gets and sets the terminal type used for a remote login session.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
UserName	Gets and sets the username used to authenticate the client session.
Version	Gets a value which returns the current version of the SshClient class library.

See Also

SshClient Class | SocketTools Namespace

SshClient.AutoResolve Property

Gets and sets a value that determines if host names and addresses are automatically resolved.

[Visual Basic]

```
Public Property AutoResolve As Boolean
```

[C#]

```
public bool AutoResolve {get; set;}
```

Property Value

Returns **true** if host names are automatically resolved to Internet addresses. The default value is **false**.

Remarks

Setting the **AutoResolve** property determines if the class automatically resolves host names and addresses specified by the **HostName** and **HostAddress** properties. If set to **true**, setting the **HostName** property will cause the class to automatically determine the corresponding IP address and update the **HostAddress** property accordingly. Likewise, setting the **HostAddress** property will cause the class to determine the host name and update the **HostName** property. Setting this property to **false** prevents the class from resolving host names until a connection attempt is made.

It is important to note that setting the **HostName** or **HostAddress** property may cause the current thread to block, sometimes for several seconds, until the name or address is resolved. To prevent this behavior, set this property value to **false**.

See Also

SshClient Class | SocketTools Namespace

SshClient.Blocking Property

Gets and sets a value which indicates if the client is in blocking mode.

[Visual Basic]

```
Public Property Blocking As Boolean
```

[C#]

```
public bool Blocking {get; set;}
```

Property Value

Returns **true** if the client is in blocking mode; otherwise it returns **false**. The default value is **true**.

Remarks

Setting the **Blocking** property determines if client operations complete synchronously or asynchronously. If set to **true**, then each client operation (such as sending or receiving data) will return when the operation has completed or timed-out. If set to **false**, client operations will return immediately. If the operation would result in the client blocking (such as attempting to read data when no data has been sent by the remote host), an error is generated.

It is important to note that certain events, such as **OnDisconnect, OnRead** and **OnWrite** are only fired if the client is in non-blocking mode.

See Also

SshClient Class | SocketTools Namespace

SshClient.CipherStrength Property

Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.

[Visual Basic]
```
Public ReadOnly Property CipherStrength As Integer
```

[C#]
```
public int CipherStrength {get;}
```

Property Value

An integer value which specifies the encryption key length if a secure connection has been established; otherwise a value of 0 is returned.

Remarks

The **CipherStrength** property returns the number of bits in the key used to encrypt the secure data stream. Common values returned by this property are 128 and 256. A key length of 40 or 56 bits is considered insecure and subject to brute force attacks. 128-bit and 256-bit keys are considered secure. If this property returns a value of 0, this means that a secure connection has not been established with the remote host.

See Also

SshClient Class | SocketTools Namespace

SshClient.CodePage Property

Gets and sets the code page used when reading and writing text.

[Visual Basic]

```
Public Property CodePage As Integer
```

[C#]

```
public int CodePage {get; set;}
```

Property Value

An integer value which specifies the current code page. A value of zero specifies the default code page for the current locale should be used. To preserve the original Unicode text, you can use code page 65001 which specifies UTF-8 character encoding.

Remarks

All data which is exchanged over a socket is sent and received as 8-bit bytes, typically referred to as "octets" in networking terminology. However, strings in .NET are Unicode where each character is represented by 16 bits. To send and receive data using strings, these Unicode strings are converted to a stream of bytes.

By default, strings are converted to an array of bytes using the code page for the current locale, mapping the 16-bit Unicode characters to bytes. Similarly, when reading data from the socket into a string buffer, the stream of bytes received from the remote host are converted to Unicode before they are returned to your application.

If you are exchanging text with another system and it appears to corrupted or characters are being replaced with question marks or other symbols, it is likely the system is sending text which is using a different character encoding. Most services use UTF-8 encoding to represent non-ASCII characters and selecting the UTF-8 code page will typically resolve the issue.

Strings are only guaranteed to be safe when sending and receiving text. Using a string data type is not recommended when reading or writing binary data to a socket. If possible, you should always use a byte array as the buffer parameter for the Read and Write methods whenever you are exchanging binary data.

For backwards compatibility, this class defaults to using the code page for the current locale. This property value directly corresponds to Windows code page identifiers, and will accept any valid code page supported by the .NET Framework. Setting this property to an invalid code page will generate an exception.

See Also

SshClient Class | SocketTools Namespace

SshClient.Columns Property

Gets and sets the number of columns for the virtual terminal session.

[Visual Basic]
```
Public Property Columns As Integer
```

[C#]
```
public int Columns {get; set;}
```

Property Value

An integer value that specifies the number of character columns for the virtual display.

Remarks

The **Columns** property returns the number of character columns for the virtual display. Setting this property prior to calling the **Connect** method requests that the server create a pseudoterminal with the specified number of columns. This property value is only meaningful for interactive terminal sessions, and is not used when executing a command on the remote host.

See Also

SshClient Class | SocketTools Namespace

SshClient.Command Property

Gets and sets the command that will be executed on the server.

[Visual Basic]

```
Public Property Command As String
```

[C#]

```
public string Command {get; set;}
```

Property Value

A string value that specifies the command which will be executed on the server.

Remarks

The **Command** property is used to specify a command and its arguments that should be executed on the server. The output of the command will be returned to the application and can be read using the **Read** or **ReadLine** method. If no command is specified, then the control will establish an interactive terminal session instead.

The command and its arguments must follow the conventions used by the SSH server, and the command will execute in the context of the authenticated user. The **ExitCode** property can be used to obtain the numerical exit code of the remote program, if one is available.

See Also

SshClient Class | SocketTools Namespace

SshClient.Fingerprint Property

Returns a string that uniquely identifies the server.

[Visual Basic]

```
Public ReadOnly Property Fingerprint As String
```

[C#]

```
public string Fingerprint {get;}
```

Property Value

A string which represents an MD5 hash value that can be used to uniquely identify the server.

Remarks

The **Fingerprint** property returns a string that consists of a series of hexadecimal values separated by colons. The value is unique to the server, and is an MD5 hash of the RSA host key. An application can use this value to determine if a connection has been established with the server previously by storing the server's host name, IP address and fingerprint in a file, registry key or a database.

See Also

SshClient Class | SocketTools Namespace

SshClient.Handle Property

Gets a value that specifies the client handle allocated for the current session.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer which represents a client handle. If there is no active connection, a value of -1 is returned.

Remarks

The **Handle** property specifies the numeric descriptor of the current client session.

See Also

SshClient Class | SocketTools Namespace

SshClient.HashStrength Property

Gets a value which specifies the length of the message digest that was selected for a secure connection.

[Visual Basic]

```
Public ReadOnly Property HashStrength As Integer
```

[C#]

```
public int HashStrength {get;}
```

Property Value

An integer value which specifies the length of the message digest if a secure connection has been established; otherwise a value of 0 is returned.

Remarks

The **HashStrength** property returns the number of bits used in the message digest (hash) that was selected. Common values returned by this property are 128 and 160. If this property returns a value of 0, this means that a secure connection has not been established with the remote host.

See Also

SshClient Class | SocketTools Namespace

SshClient.HostAddress Property

Gets and sets a value which specifies the Internet address used to establish a connection.

[Visual Basic]

```
Public Property HostAddress As String
```

[C#]

```
public string HostAddress {get; set;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **HostAddress** property can be used to set the Internet address for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the address is assigned to a valid host name, the **HostName** property will be updated with that value.

See Also

SshClient Class | SocketTools Namespace

SshClient.HostName Property

Gets and sets a value which specifies the host name used to establish a connection.

[Visual Basic]

```
Public Property HostName As String
```

[C#]

```
public string HostName {get; set;}
```

Property Value

A string which specifies a host name.

Remarks

The **HostName** property can be used to set the host name for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the name can be resolved to a valid Internet address, the **HostAddress** property will be updated with that value.

See Also

SshClient Class | SocketTools Namespace

SshClient.IsBlocked Property

Gets a value which indicates if the current thread is performing a blocking client operation.

[Visual Basic]

```
Public ReadOnly Property IsBlocked As Boolean
```

[C#]

```
public bool IsBlocked {get;}
```

Property Value

Returns **true** if the current thread is blocking, otherwise returns **false**.

Remarks

The **IsBlocked** property returns **true** if the current thread is blocked performing an operation. Because the Windows Sockets API only permits one blocking operation per thread of execution, this property should be checked before starting any blocking operation in response to an event.

If the **IsBlocked** property returns **false**, this means there are no blocking operations on the current thread at that time. However, this does not guarantee that the next client operation will not fail. An application should always check the return value from a client operation and check the value of the **LastError** property if an error occurs.

Note that this property will return **true** if there is any blocking operation being performed by the current thread, regardless of whether this specific instance of the class is responsible for the blocking operation or not.

See Also

SshClient Class | SocketTools Namespace

SshClient.IsConnected Property

Gets a value which indicates if a connection to the remote host has been established.

[Visual Basic]

```
Public ReadOnly Property IsConnected As Boolean
```

[C#]

```
public bool IsConnected {get;}
```

Property Value

Returns **true** if the connection has been established; otherwise returns **false**.

Remarks

The **IsConnected** property can only be used to indicate if there is still a logical connection to the remote host. It cannot be used to detect abnormal conditions such as the remote host aborting the connection, the physical network connection being lost or other critical errors.

See Also

SshClient Class | SocketTools Namespace

SshClient.IsInitialized Property

Gets a value which indicates if the current instance of the class has been initialized successfully.

[Visual Basic]

```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]

```
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the class instance has been initialized; otherwise returns **false**.

Remarks

The **IsInitialized** property is used to determine if the current instance of the class has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the class may not be able to initialize itself.

The most common reasons that a class instance may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required networking libraries or not being able to access the system registry.

See Also

SshClient Class | SocketTools Namespace

SshClient.IsReadable Property

Gets a value which indicates if there is data available to be read from the socket connection to the server.

[Visual Basic]

```
Public ReadOnly Property IsReadable As Boolean
```

[C#]

```
public bool IsReadable {get;}
```

Property Value

Returns **true** if there is data available to be read; otherwise returns **false**.

Remarks

The **IsReadable** property returns **true** if data can be read from the client without blocking. For non-blocking sessions, this property can be checked before the application attempts to read the client. Note that even if this property does return **true** indicating that there is data available to be read, applications should always check the return value from the **Read** method.

See Also

SshClient Class | SocketTools Namespace

SshClient.IsWritable Property

Gets a value which indicates if data can be written to the client without blocking.

[Visual Basic]

```
Public ReadOnly Property IsWritable As Boolean
```

[C#]

```
public bool IsWritable {get;}
```

Property Value

Returns **true** if data can be written to the client; otherwise returns **false**.

Remarks

The **IsWritable** property returns **true** if data can be written to the client without blocking. For non-blocking sessions, this property can be checked before the application attempts to write data to the client. Note that even if this property does return **true** indicating that data can be written to the client, applications should always check the return value from the **Write** method.

See Also

SshClient Class | SocketTools Namespace

SshClient.KeepAlive Property

Gets and sets a value which determines if the client session should kept active.

[Visual Basic]

```
Public Property KeepAlive As Boolean
```

[C#]

```
public bool KeepAlive {get; set;}
```

Property Value

A boolean value which specifies if the client connection should be maintained over a long period of time. A value of **true** indicates that the client should attempt to keep the connection active, even if it is idle for more than two hours. A value of **false** specifies that the connection should be established normally.

Remarks

Setting the **KeepAlive** property to a value of true indicates that the client wishes to maintain a long-duration session with the server. It is only necessary to set this property to a value of true if the client session is interactive and the connection must be held open for more than two hours.

See Also

SshClient Class | SocketTools Namespace

SshClient.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public SshClient.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

SshClient Class | SocketTools Namespace

SshClient.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

SshClient Class | SocketTools Namespace

SshClient.LocalAddress Property

Gets the local Internet address that the client is bound to.

[Visual Basic]

```
Public ReadOnly Property LocalAddress As String
```

[C#]

```
public string LocalAddress {get;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **LocalAddress** property returns the local Internet address that the client is bound to when a connection is established with a remote host. This property may return either an IPv4 or IPv6 formatted address, depending on the type of connection that was established.

See Also

SshClient Class | SocketTools Namespace

SshClient.LocalName Property

Gets a value which specifies the host name for the local system.

[Visual Basic]
```
Public ReadOnly Property LocalName As String
```

[C#]
```
public string LocalName {get;}
```

Property Value

A string which specifies the local host name.

Remarks

The **LocalName** property returns the fully-qualified host name assigned to the local system. If the system has not been configured with an Internet domain name, then this property will return the NetBIOS name assigned to the local system.

See Also

SshClient Class | SocketTools Namespace

SshClient.LocalPort Property

Gets the local port number the client is bound to.

[Visual Basic]
```
Public ReadOnly Property LocalPort As Integer
```

[C#]
```
public int LocalPort {get;}
```

Property Value

An integer value which specifies a port number. The default value is 0.

Remarks

The **LocalPort** property is used to identify the local port number that the client is bound to to when a connection is established with a remote host.

See Also

SshClient Class | SocketTools Namespace

SshClient.NewLine Property

Gets and sets the end-of-line character sequences sent to the server.

[Visual Basic]
```
Public Property NewLine As LineMode
```

[C#]
```
public SshClient.LineMode NewLine {get; set;}
```

Property Value

Returns an LineMode enumeration value which specify the current line mode for the client. The default value for this property is **newLineDefault**.

Remarks

When a connection is initially established with the server, it determines what characters are used to indicate the end-of-line and how they are displayed. On UNIX based systems, this is controlled by the settings for the pseudo-terminal that is allocated for the client session, and can be changed using the **stty** command. In most cases, the client line mode can be left at the default. However, in some cases you may need to change the line mode, particularly if you intend to send data from a Windows text file or copied from the clipboard.

Windows uses a carriage return and linefeed (CRLF) sequence to indicate the end-of-line and a UNIX based server may interpret that as multiple newlines. To prevent this, set the **NewLine** property to **LineMode.newLineCR** and the CRLF sequence in the text will be replaced by a single carriage return.

See Also

SshClient Class | SocketTools Namespace | Write Method | WriteLine Method

SshClient.Options Property

Gets and sets a value which specifies one or more client options.

[Visual Basic]

```
Public Property Options As ClientOptions
```

[C#]

```
public SshClient.ClientOptions Options {get; set;}
```

Property Value

Returns one or more ClientOptions enumeration flags which specify the options for the client. The default value for this property is **clientOptionNone**.

Remarks

The **Options** property specifies one or more default options options which are used when establishing a connection using the **Connect** method.

See Also

SshClient Class | SocketTools Namespace

SshClient.Password Property

Gets and sets the password used to authenticate the client session.

[Visual Basic]

```
Public Property Password As String
```

[C#]

```
public string Password {get; set;}
```

Property Value

A string which specifies the password.

Remarks

If a password is not specified when the **Login** method is called, the value of this property will be used as the default password when authenticating the client session.

See Also

SshClient Class | SocketTools Namespace

SshClient.PrivateKey Property

Gets and sets the name of the private key file used to authenticate the client session.

[Visual Basic]

```
Public Property PrivateKey As String
```

[C#]

```
public string PrivateKey {get; set;}
```

Property Value

A string value which specifies the name of the private key file. An empty string specifies that no private key is required to establish a connection with the server.

Remarks

The **PrivateKey** property specifies the name of the file that contains the private key that is used to authenticate the client session. It is only necessary to set this property if the server requires the user to provide a private key to establish the connection.

See Also

SshClient Class | SocketTools Namespace

SshClient.ProxyHost Property

Gets and sets the hostname or IP address of a proxy server.

[Visual Basic]

```
Public Property ProxyHost As String
```

[C#]

```
public string ProxyHost {get; set;}
```

Property Value

A string which specifies the hostname or IP address of the proxy server that will be used when establishing a connection.

Remarks

The **ProxyHost** property should be set to the name of the proxy server that you want to connect to. This property may be set to either a fully qualified domain name, or an IP address. This property is only used if the **ProxyType** property specifies a proxy server type other than **proxyNone**.

See Also

SshClient Class | SocketTools Namespace

SshClient.ProxyPassword Property

Gets and sets the password used to authenticate the connection to a proxy server.

[Visual Basic]

```
Public Property ProxyPassword As String
```

[C#]

```
public string ProxyPassword {get; set;}
```

Property Value

A string which specifies a password.

Remarks

The **ProxyPassword** property specifies the password used to authenticate the user to the proxy server. If a password is not required by the server, this property is ignored.

See Also

SshClient Class | SocketTools Namespace

SshClient.ProxyPort Property

Gets and sets a value that specifies the proxy server port number.

[Visual Basic]

```
Public Property ProxyPort As Integer
```

[C#]

```
public int ProxyPort {get; set;}
```

Property Value

An integer value which specifies the proxy port number.

Remarks

The **ProxyPort** property is used to set the port number that the control will use to establish a connection with the proxy server. A value of zero specifies that the client will connect to the proxy server using the standard HTTP service port.

See Also

SshClient Class | SocketTools Namespace

SshClient.ProxyType Property

Gets and sets the type of proxy server the client will use to establish a connection.

[Visual Basic]

```
Public Property ProxyType As ProxyTypes
```

[C#]

```
public SshClient.ProxyTypes ProxyType {get; set;}
```

Property Value

An ProxyTypes enumeration which specifies the type of proxy that the client will connect through.

See Also

SshClient Class | SocketTools Namespace

SshClient.ProxyUser Property

Gets and sets the username used to authenticate the connection to a proxy server.

[Visual Basic]

```
Public Property ProxyUser As String
```

[C#]

```
public string ProxyUser {get; set;}
```

Property Value

A string which specifies the username.

Remarks

The **ProxyUser** property specifies the user that is logging in to the proxy server. If the proxy server does not require the user to login, then this property is ignored.

See Also

SshClient Class | SocketTools Namespace

SshClient.RemotePort Property

Gets and sets a value which specifies the remote port number.

[Visual Basic]

```
Public Property RemotePort As Integer
```

[C#]

```
public int RemotePort {get; set;}
```

Property Value

An integer value which specifies a port number.

Remarks

The **RemotePort** property is used to set the port number that will be used to establish a connection with a remote host. If the port number specifies a well-known port, the **RemoteService** property will be updated with that name.

See Also

SshClient Class | SocketTools Namespace

SshClient.Rows Property

Gets and sets the number of rows for the virtual terminal session.

[Visual Basic]

```
Public Property Rows As Integer
```

[C#]

```
public int Rows {get; set;}
```

Property Value

An integer value that specifies the number of character rows for the virtual display.

Remarks

The **Rows** property returns the number of character rows for the virtual display. Setting this property prior to calling the **Connect** method requests that the server create a pseudoterminal with the specified number of rows. This property value is only meaningful for interactive terminal sessions, and is not used when executing a command on the remote host.

The default value for this property is 24.

See Also

SshClient Class | SocketTools Namespace

SshClient.Secure Property

Gets and sets a value which specifies if a secure connection is established.

[Visual Basic]

```
Public Property Secure As Boolean
```

[C#]

```
public bool Secure {get; set;}
```

Property Value

Returns **true** if a secure connection is established; otherwise returns **false**. The default value is **false**.

Remarks

The **Secure** property determines if a secure connection is established to the server. The default value for this property is true, and it is included only for compatibility with the other SocketTools components. Because all SSH connections must be secure, attempting to set this property to a value of false will result in an error.

See Also

SshClient Class | SocketTools Namespace

SshClient.SecureCipher Property

Gets a value that specifies the encryption algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureCipher As SecureCipherAlgorithm
```

[C#]

```
public SshClient.SecureCipherAlgorithm SecureCipher {get;}
```

Property Value

A SecureCipherAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureCipher** property returns a value which identifies the algorithm used to encrypt the data stream. If a secure connection has not been established, this property will return a value of **cipherNone**.

See Also

SshClient Class | SocketTools Namespace

SshClient.SecureHash Property

Gets a value that specifies the message digest algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureHash As SecureHashAlgorithm
```

[C#]

```
public SshClient.SecureHashAlgorithm SecureHash {get;}
```

Property Value

A SecureHashAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureHash** property returns a value which identifies the message digest (hash) algorithm that was selected when a secure connection was established. If a secure connection has not been established, this property will return a value of **hashNone**.

See Also

SshClient Class | SocketTools Namespace

SshClient.SecureKeyExchange Property

Gets a value that specifies the key exchange algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureKeyExchange As SecureKeyAlgorithm
```

[C#]

```
public SshClient.SecureKeyAlgorithm SecureKeyExchange {get;}
```

Property Value

A SecureKeyAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureKeyExchange** property returns a value which identifies the key exchange algorithm that was selected when a secure connection was established. If a secure connection has not been established, this property will return a value of **keyExchangeNone**.

See Also

SshClient Class | SocketTools Namespace

SshClient.SecureProtocol Property

Gets and sets a value which specifies the protocol used for a secure connection.

[Visual Basic]
```
Public Property SecureProtocol As SecurityProtocols
```

[C#]
```
public SshClient.SecurityProtocols SecureProtocol {get; set;}
```

Property Value

A SecurityProtocols enumeration value which identifies the protocol to be used when establishing a secure connection.

Remarks

The **SecureProtocol** property can be used to specify the security protocol to be used when establishing a secure connection with a server. By default, the control will attempt to use either SSH-1 or SSH-2 to establish the connection, with the appropriate protocol automatically selected based on the capabilities of the server. It is recommended that you only change this property value if you fully understand the implications of doing so. Assigning a value to this property will override the default and force the class to attempt to use only the protocol specified.

Multiple security protocols may be specified by combining the SecurityProtocols using a bitwise Or operator. After a connection has been established, reading this property will identify the protocol that was selected to establish the connection. Attempting to set this property after a connection has been established will result in an exception being thrown. This property should only be set before calling the **Connect** method.

See Also

SshClient Class | SocketTools Namespace

SshClient.Status Property

Gets a value which specifies the current status of the client.

[Visual Basic]

```
Public ReadOnly Property Status As ClientStatus
```

[C#]

```
public SshClient.ClientStatus Status {get;}
```

Property Value

A ClientStatus enumeration value which specifies the current client status.

Remarks

The **Status** property returns the current status of the client. This property can be used to check on blocking connections to determine if the client is interacting with the remote host before taking some action.

See Also

SshClient Class | SocketTools Namespace

SshClient.Terminal Property

Gets and sets the terminal type used for a remote login session.

[Visual Basic]

```
Public Property Terminal As String
```

[C#]

```
public string Terminal {get; set;}
```

Property Value

A string which specifies the terminal type.

Remarks

The **Terminal** property specifies the terminal type of the remote host for display purposes. On UNIX based systems, the terminal name corresponds to a termcap or terminfo entry as set in the TERM environment variable. On Windows based systems which implement the rlogin service, this property may be ignored and the server will assume that the client is capable of displaying ANSI escape sequences. On VMS systems, the terminal name should correspond to the terminal type used with the SET TERMINAL/DEVICE command.

If this property is set to an empty string and no terminal type is specified when the **Login** method is called, a default terminal type named "unknown" will be used. On most UNIX and VMS systems this defines a terminal which is not capable of cursor positioning using control or escape sequences. This terminal type may not be recognized and an error may be displayed when the user logs in indicating that the terminal type is invalid.

Refer to the documentation for the server system to determine what terminal type names are available to you. Remember that on UNIX systems, the terminal type is case-sensitive. Some of the more common terminal types are:

Terminal	Description
ansi	This terminal type is usually available on UNIX based servers. This specifies that the client is capable of displaying standard ANSI escape sequences for cursor control.
dumb	This terminal type typically specifies a terminal display which does not support control or escape sequences for cursor positioning. If you do not want escape sequences embedded in the data stream and the server returns an error if the terminal type is not specified, try using this terminal type.
pcansi	This terminal type is usually available on UNIX based servers. This specifies that the client is a using a PC terminal emulator that supports basic ANSI escape sequences for cursor control. This may also enable escape sequences which can set the display colors.
vt100	This terminal type is usually available on UNIX and VMS based servers. On some VMS systems this

	string may need to be specified as DEC-VT100. This specifies that the client is capable of emulating a DEC VT100 terminal. The VT100 supports many of the same cursor control sequences as an ANSI terminal.
vt220	This terminal type is usually available on UNIX and VMS based servers. On some VMS systems this string may need to be specified as DEC-VT220. This specifies that the client is capable of emulating a DEC VT220 terminal, which is a later version of the VT100.
vt320	This terminal type is usually available on UNIX and VMS based servers. On some VMS systems this string may need to be specified as DEC-VT320. This specifies that the client is capable of emulating a DEC VT320 terminal, which is similar to the VT100 and VT220 and provides advanced features such as the ability to set display colors.
xterm	This terminal type is may be available on UNIX based servers which have X Windows installed. This specifies that the client is a using the X Windows xterm emulator which supports standard ANSI escape sequences for cursor control.

See Also

SshClient Class | SocketTools Namespace

SshClient.ThreadModel Property

Gets and sets a value which specifies the threading model for the class instance.

[Visual Basic]

```
Public Property ThreadModel As ThreadingModel
```

[C#]

```
public SshClient.ThreadingModel ThreadModel {get; set;}
```

Property Value

Returns one or more ThreadingModel enumeration value which specifies the threading model for the client. The default value for this property is **modelSingleThread**.

Remarks

The **ThreadModel** property specifies the threading model that is used by the class instance when a connection is established. The default value for this property is **modelSingleThread**, which specifies that only the thread that established the connection should be permitted to invoke methods. It is important to note that this threading model does not limit the application to a single thread of execution. When a session is established using the **Connect** method, that session is attached to the thread that created it. From that point on, until the session is terminated, only the owner may invoke methods in that instance of the class. The ownership of the class instance may be transferred from one thread to another using the **AttachThread** method.

Setting this property to **modelFreeThread** disables certain internal safety checks that are performed by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads. The application must ensure that no two threads will attempt to invoke a blocking method at the same time. In other words, if one thread invokes a method, the application must ensure that another thread will not attempt to invoke any other method at the same time using the same instance of the class.

Changing the value of this property will not affect an active client session. The threading model must be specified prior to invoking the **Connect** method.

See Also

SshClient Class | SocketTools Namespace | AttachThread Method | ThreadingModel Enumeration | ThreadModel Attribute

SshClient.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]

```
Public Property ThrowError As Boolean
```

[C#]

```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

SshClient Class | SocketTools Namespace

SshClient.Timeout Property

Gets and sets a value which specifies a timeout period in seconds.

[Visual Basic]

```
Public Property Timeout As Integer
```

[C#]

```
public int Timeout {get; set;}
```

Property Value

An integer value which specifies a timeout period in seconds.

Remarks

Setting the **Timeout** property specifies the number of seconds until a blocking operation fails and returns an error.

The timeout period is only used when the client is in blocking mode. Although this property can be changed when the client is in non-blocking mode, the value will be ignored until the client is returned to blocking mode.

For most applications it is recommended the timeout period be set between 10 and 20 seconds.

See Also

SshClient Class | SocketTools Namespace

SshClient.Trace Property

Gets and sets a value which indicates if network function logging is enabled.

[Visual Basic]
```
Public Property Trace As Boolean
```

[C#]
```
public bool Trace {get; set;}
```

Property Value

Returns **true** if network function tracing is enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Trace** property is used to enable (or disable) the tracing of network function calls. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the SocketTools networking classes and is enabled or disabled for an entire process. This means that once trace logging is enabled for a given component, all of the function calls made by the process using any of the SocketTools classes will be logged. For example, if you have an application using both the File Transfer Protocol and Post Office Protocol classes, and you set the **Trace** property to **true**, function calls made by both classes will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all classes used by the process.

If trace logging is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the logfile is fairly large. Since logfiles can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

When redistributing your application, make sure that you include the **SocketTools10.TraceLog.dll** module with your installation. If this library is not present, then no trace output will be generated and the value of the **Trace** property will be ignored. Only those function calls made by the SocketTools networking classes will be logged. Calls made directly to the Windows Sockets API, or calls made by other classes, will not be logged.

See Also

SshClient Class | SocketTools Namespace

SshClient.TraceFile Property

Gets and sets a value which specifies the name of the logfile.

[Visual Basic]
```
Public Property TraceFile As String
```

[C#]
```
public string TraceFile {get; set;}
```

Property Value

A string which specifies the name of the file.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named **SocketTools.log** is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFile** property should be set to the same value for each instance. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
 MyApp INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0 MyApp WRN:
connect(46, 192.0.0.1:1234, 16) returned -1 [10035] MyApp ERR: accept(46,
NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the control is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Note that if the specified file cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

SshClient Class | SocketTools Namespace

SshClient.TraceFlags Property

Gets and sets a value which specifies the client function tracing flags.

[Visual Basic]

```
Public Property TraceFlags As TraceOptions
```

[C#]

```
public SshClient.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration which specifies the amount of detail written to the trace logfile.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Because network function tracing is enabled per-process, the trace flags are shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFlags** property should be set to the same value for each instance. Changing the trace flags for any one instance of the class will affect the logging performed for all SocketTools classes used by the application.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being written and the error **errorOperationWouldBlock** occurs, a warning is generated because the application simply needs to attempt to write the data at a later time.

See Also

SshClient Class | SocketTools Namespace

SshClient.UserName Property

Gets and sets the username used to authenticate the client session.

[Visual Basic]

```
Public Property UserName As String
```

[C#]

```
public string UserName {get; set;}
```

Property Value

A string which specifies the username.

Remarks

If a username is not specified when the **Login** method is called, the value of this property will be used as the default username when authenticating the client session.

See Also

SshClient Class | SocketTools Namespace

SshClient.Version Property

Gets a value which returns the current version of the SshClient class library.

[Visual Basic]

```
Public ReadOnly Property Version As String
```

[C#]

```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the SshClient class library. This value can be used by an application for validation and debugging purposes.

See Also

SshClient Class | SocketTools Namespace

SshClient Methods

The methods of the **SshClient** class are listed below. For a complete list of **SshClient** class members, see the SshClient Members topic.

Public Instance Methods

AttachThread	Attach an instance of the class to the current thread
Break	Sends a break signal to the remote host.
Cancel	Cancel the current blocking client operation.
Connect	Overloaded. Establish a connection with a remote host.
Control	Send a control message to the server.
Disconnect	Terminate the connection with a remote host.
Dispose	Overloaded. Releases all resources used by SshClient.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
Execute	Overloaded. Execute a command on the server and return the output.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the SshClient class.
Peek	Overloaded. Read data returned by the server, but do not remove it from the receive buffer.
Read	Overloaded. Read data from the server and store it in a byte array.
ReadLine	Overloaded. Read up to a line of data from the server and return it in a string buffer.
Reset	Reset the internal state of the object, resetting all properties to their default values.
Search	Overloaded. Search for a specific character sequence in the data stream.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
Write	Overloaded. Write one or more bytes of data to the server.

WriteLine	Overloaded. Send a line of text to the server, terminated by a carriage-return and linefeed.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the SshClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

SshClient Class | SocketTools Namespace

SshClient.AttachThread Method

Attach an instance of the class to the current thread

[Visual Basic]
```
Public Function AttachThread() As Boolean
```

[C#]
```
public bool AttachThread();
```

Return Value

A boolean value which specifies if the client could be attached to the current thread. If this method returns **false**, the client could not be attached to the thread and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

When an instance of the class is created it is associated with the current thread that created it. Normally, if another thread attempts to perform an operation using that instance, an error is returned. This is used to ensure that other threads cannot interfere with an operation being performed by the owner thread. In some cases, it may be desirable for one thread in an application to create an instance of the class, establish a connection and then pass that instance to another worker thread. The **AttachThread** method can be used to change the ownership of the class instance to the new worker thread.

This method should be called by the new thread immediately after it has been created, and if the new thread does not release the handle itself, the ownership of the handle should be restored by the original thread. Under no circumstances should **AttachThread** be used to forcibly destroy an instance of a class allocated by another thread while a blocking operation is in progress. To cancel a blocking operation, use the **Cancel** method and then delete the class instance after the blocking function exits and control is returned to the current thread.

See Also

SshClient Class | SocketTools Namespace

SshClient.Break Method

Sends a break signal to the remote host.

[Visual Basic]

```
Public Function Break() As Boolean
```

[C#]

```
public bool Break();
```

Return Value

This method returns a Boolean value which specifies if the break signal was sent to the server. A value of **true** indicates the method was successful. If an error occurs, the method will return **false**.

Remarks

The **Break** method a control message to the remote host which simulates a break signal on a physical terminal. This is used by some operating systems as an instruction to enter a privileged configuration mode. Note that this is not the same as sending an interrupt character such as Ctrl+C to the remote host. This method is ignored for SSH 1.0 sessions.

See Also

SshClient Class | SocketTools Namespace

SshClient.Cancel Method

Cancel the current blocking client operation.

[Visual Basic]

```
Public Sub Cancel()
```

[C#]

```
public void Cancel();
```

Remarks

When the **Cancel** method is called, the blocking client operation will not immediately fail. An internal flag is set which causes the blocking operation to exit with an error. This means that the application cannot cancel an operation and immediately perform some other blocking function. Instead it must allow the calling stack to unwind, returning back to the blocking operation before making any further function calls.

See Also

SshClient Class | SocketTools Namespace

SshClient.Connect Method

Establish a connection with a remote host.

Overload List

Establish a connection with a remote host.

public bool Connect();

Establish a connection with a remote host.

public bool Connect(string);

Establish a connection with a remote host.

public bool Connect(string,int);

Establish a connection with a remote host.

public bool Connect(string,int,int);

Establish a connection with a remote host.

public bool Connect(string,int,string,string);

Establish a connection with a remote host.

public bool Connect(string,int,string,string,int,ClientOptions);

Establish a connection with a remote host.

public bool Connect(string,string,string);

See Also

SshClient Class | SocketTools Namespace

SshClient.Connect Method ()

Establish a connection with a remote host.

[Visual Basic]
```
Overloads Public Function Connect() As Boolean
```

[C#]
```
public bool Connect();
```

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection using assigned property values.

The value of the **HostName** or **HostAddress** property will be used to determine the host name or address to connect to.

The value of the **RemotePort** property will be used to determine the port number to connect to.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

See Also

SshClient Class | SocketTools Namespace | SshClient.Connect Overload List

SshClient.Connect Method (String)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection by using assigned property values.

The value of the **RemotePort** property will be used to determine the port number to connect to.

The value of the **UserName** and **Password** properties will be used to authenticate the session.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

See Also

SshClient Class | SocketTools Namespace | SshClient.Connect Overload List

SshClient.Connect Method (String, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection by using assigned property values.

The value of the **UserName** and **Password** properties will be used to authenticate the session.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

See Also

SshClient Class | SocketTools Namespace | SshClient.Connect Overload List

SshClient.Connect Method (String, Int32, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   int timeout
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection by using assigned property values.

The value of the **UserName** and **Password** properties will be used to authenticate the session.

The value of the **Options** property will be used to specify the default options for the connection.

See Also

SshClient Class | SocketTools Namespace | SshClient.Connect Overload List

SshClient.Connect Method (String, Int32, String, String)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal password As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string password
);
```

Parameters

hostName
A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort
An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName
A string which specifies the user name which will be used to authenticate the client session.

password
A string which specifies the password which will be used to authenticate the client session.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection by using assigned property values.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

See Also

SshClient Class | SocketTools Namespace | SshClient.Connect Overload List

SshClient.Connect Method (String, Int32, String, String, Int32, ClientOptions)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal password As String, _
   ByVal timeout As Integer, _
   ByVal options As ClientOptions _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   string userName,
   string password,
   int timeout,
   ClientOptions options
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName

A string which specifies the user name which will be used to authenticate the client session.

password

A string which specifies the password which will be used to authenticate the client session.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

options

One or more of the ClientOptions enumeration flags.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

SshClient Class | SocketTools Namespace | SshClient.Connect Overload List

SshClient.Connect Method (String, String, String)

Establish a connection with a remote host.

```
[Visual Basic]
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal userName As String, _
   ByVal password As String _
) As Boolean
```

```
[C#]
public bool Connect(
   string hostName,
   string userName,
   string password
);
```

Parameters

hostName
A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

userName
A string which specifies the user name which will be used to authenticate the client session.

password
A string which specifies the password which will be used to authenticate the client session.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection by using assigned property values.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

See Also

SshClient Class | SocketTools Namespace | SshClient.Connect Overload List

SshClient.Control Method

Send a control message to the server.

```
[Visual Basic]
Public Function Control( _
   ByVal code As ControlCodes _
) As Boolean
```

```
[C#]
public bool Control(
   ControlCodes code
);
```

Parameters

code

A numeric control code which specifies the control message which should be sent to the server. This argument must specify one of the ControlCodes enumeration values.

Return Value

This method returns a Boolean value which specifies if the control code was sent to the server. A value of **true** indicates the method was successful. If an error occurs, the method will return **false**.

Remarks

The **Control** method enables an application to send control messages to the server, which can cause it to take specific actions such as simulate a terminal break or request that the key exchange be performed again. Some control messages are not supported by the SSH 1.0 protocol, in which case the control message is ignored.

See Also

SshClient Class | SocketTools Namespace

SshClient.Disconnect Method

Terminate the connection with a remote host.

[Visual Basic]

```
Public Sub Disconnect()
```

[C#]

```
public void Disconnect();
```

Remarks

The **Disconnect** method terminates the connection with the remote host and releases the client handle allocated by the class. Note that the client socket is not immediately released when the connection is terminated and will enter a wait state for two minutes. After the time wait period has elapsed, the client will be released by the operating system. This is a normal safety mechanism to handle any packets that may arrive after the connection has been closed.

The value of the **ExitCode** property is updated when the **Disconnect** method is called. This enables an application to check the exit code returned by the shell or program that was executed on the server. In most cases, an exit code value of zero indicates success, while any other value indicates an error condition.

See Also

SshClient Class | SocketTools Namespace

SshClient.Dispose Method

Releases all resources used by SshClient.

Overload List

Releases all resources used by SshClient.

public void Dispose();

Releases the unmanaged resources allocated by the SshClient class and optionally releases the managed resources.

protected virtual void Dispose(bool);

See Also

SshClient Class | SocketTools Namespace

SshClient.Dispose Method ()

Releases all resources used by SshClient.

[Visual Basic]

```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]

```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

SshClient Class | SocketTools Namespace | SshClient.Dispose Overload List

SshClient.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the SshClient class and optionally releases the managed resources.

[Visual Basic]

```
Overridable Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]

```
protected virtual void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **SshClient** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

SshClient Class | SocketTools Namespace | SshClient.Dispose Overload List

SshClient.Execute Method

Execute a command on the server and return the output.

Overload List

Execute a command on the server and return the output.

public string Execute();

Execute a command on the server and return the output.

public string Execute(string);

Execute a command on the server and return the output.

public string Execute(string,int);

Execute a command on the server and return the output.

public string Execute(string,int,string);

Execute a command on the server and return the output.

public string Execute(string,int,string,int);

Execute a command on the server and return the output.

public string Execute(string,int,string,string,string);

Execute a command on the server and return the output.

public string Execute(string,int,string,string,string,int);

Execute a command on the server and return the output.

public string Execute(string,int,string,string,string,int,ClientOptions);

Execute a command on the server and return the output.

public string Execute(string,string);

See Also

SshClient Class | SocketTools Namespace

SshClient.Execute Method ()

Execute a command on the server and return the output.

[Visual Basic]
```
Overloads Public Function Execute() As String
```

[C#]
```
public string Execute();
```

Return Value

A string that contains the output of the command that was executed on the server. To get the exit code returned by the program, check the value of the **ExitCode** property. If an empty string is returned, this indicates that there was either no data available, or an error has occurred and the **LastError** property will return a non-zero value.

Remarks

The **Execute** method establishes a network connection with a remote server and executes the specified command. The output from the command is returned as a string. This method uses the following property values:

The value of the **HostName** or **HostAddress** property will be used to determine the server hostname or IP address.

The value of the **RemotePort** property will be used to determine the port number to connect to.

The value of the **UserName** and **Password** properties will be used to authenticate the session.

The value of the **Command** property will be used to specify the command that should executed on the server.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

This method should not be used if the connection to the server must be established through a proxy server. If the connection must be made through a proxy server, then you should set the **Command** property to the specify the command to execute, call the **Connect** method to establish the connection, and then use either the **Read** or **ReadLine** methods to read the output.

When the command output is being read from the server, this method will automatically convert the data to match the end-of-line convention used on the Windows platform. This is useful when executing a command on a UNIX based system where the end-of-line is indicated by a single linefeed, while on Windows it is a carriage-return and linefeed pair. If the output contains embedded nulls or escape sequences, then this conversion will not be performed.

See Also

SshClient Class | SocketTools Namespace | SshClient.Execute Overload List

SshClient.Execute Method (String)

Execute a command on the server and return the output.

[Visual Basic]
```
Overloads Public Function Execute( _
   ByVal command As String _
) As String
```

[C#]
```
public string Execute(
   string command
);
```

Parameters

command

A string which specifies the command that will be executed on the server.

Return Value

A string that contains the output of the command that was executed on the server. To get the exit code returned by the program, check the value of the **ExitCode** property. If an empty string is returned, this indicates that there was either no data available, or an error has occurred and the **LastError** property will return a non-zero value.

Remarks

The **Execute** method establishes a network connection with a remote server and executes the specified command. The output from the command is returned as a string. This method uses the following property values:

The value of the **HostName** or **HostAddress** property will be used to determine the server hostname or IP address.

The value of the **RemotePort** property will be used to determine the port number to connect to.

The value of the **UserName** and **Password** properties will be used to authenticate the session.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

This method should not be used if the connection to the server must be established through a proxy server. If the connection must be made through a proxy server, then you should set the **Command** property to the specify the command to execute, call the **Connect** method to establish the connection, and then use either the **Read** or **ReadLine** methods to read the output.

When the command output is being read from the server, this method will automatically convert the data to match the end-of-line convention used on the Windows platform. This is useful when executing a command on a UNIX based system where the end-of-line is indicated by a single linefeed, while on Windows it is a carriage-return and linefeed pair. If the output contains embedded nulls or escape sequences, then this conversion will not be performed.

See Also

SshClient Class | SocketTools Namespace | SshClient.Execute Overload List

SshClient.Execute Method (String, Int32)

Execute a command on the server and return the output.

[Visual Basic]

```
Overloads Public Function Execute( _
   ByVal command As String, _
   ByVal timeout As Integer _
) As String
```

[C#]

```
public string Execute(
   string command,
   int timeout
);
```

Parameters

command

A string which specifies the command that will be executed on the server.

timeout

The number of seconds that the client will wait for a response before failing the operation.

Return Value

A string that contains the output of the command that was executed on the server. To get the exit code returned by the program, check the value of the **ExitCode** property. If an empty string is returned, this indicates that there was either no data available, or an error has occurred and the **LastError** property will return a non-zero value.

Remarks

The **Execute** method establishes a network connection with a remote server and executes the specified command. The output from the command is returned as a string. This method uses the following property values:

The value of the **HostName** or **HostAddress** property will be used to determine the server hostname or IP address.

The value of the **RemotePort** property will be used to determine the port number to connect to.

The value of the **UserName** and **Password** properties will be used to authenticate the session.

The value of the **Options** property will be used to specify the default options for the connection.

This method should not be used if the connection to the server must be established through a proxy server. If the connection must be made through a proxy server, then you should set the **Command** property to the specify the command to execute, call the **Connect** method to establish the connection, and then use either the **Read** or **ReadLine** methods to read the output.

When the command output is being read from the server, this method will automatically convert the data to match the end-of-line convention used on the Windows platform. This is useful when executing a command on a UNIX based system where the end-of-line is indicated by a single linefeed, while on Windows it is a carriage-return and linefeed pair. If the output contains embedded nulls or escape sequences, then this conversion will not be performed.

See Also

SshClient Class | SocketTools Namespace | SshClient.Execute Overload List

SshClient.Execute Method (String, String)

Execute a command on the server and return the output.

[Visual Basic]

```
Overloads Public Function Execute( _
   ByVal hostName As String, _
   ByVal command As String _
) As String
```

[C#]

```
public string Execute(
   string hostName,
   string command
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

command

A string which specifies the command that will be executed on the server.

Return Value

A string that contains the output of the command that was executed on the server. To get the exit code returned by the program, check the value of the **ExitCode** property. If an empty string is returned, this indicates that there was either no data available, or an error has occurred and the **LastError** property will return a non-zero value.

Remarks

The **Execute** method establishes a network connection with a remote server and executes the specified command. The output from the command is returned as a string. This method uses the following property values:

The value of the **RemotePort** property will be used to determine the port number to connect to.

The value of the **UserName** and **Password** properties will be used to authenticate the session.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

This method should not be used if the connection to the server must be established through a proxy server. If the connection must be made through a proxy server, then you should set the **Command** property to the specify the command to execute, call the **Connect** method to establish the connection, and then use either the **Read** or **ReadLine** methods to read the output.

When the command output is being read from the server, this method will automatically convert the data to match the end-of-line convention used on the Windows platform. This is useful when executing a command on a UNIX based system where the end-of-line is indicated by a single linefeed, while on Windows it is a carriage-return and linefeed pair. If the output contains embedded nulls or escape sequences, then this conversion will not be performed.

See Also

SshClient Class | SocketTools Namespace | SshClient.Execute Overload List

SshClient.Execute Method (String, Int32, String, Int32)

Execute a command on the server and return the output.

```
[Visual Basic]
Overloads Public Function Execute( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal command As String, _
   ByVal timeout As Integer _
) As String
```

```
[C#]
public string Execute(
   string hostName,
   int hostPort,
   string command,
   int timeout
);
```

Parameters

hostName
A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort
An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

command
A string which specifies the command that will be executed on the server.

timeout
The number of seconds that the client will wait for a response before failing the operation.

Return Value

A string that contains the output of the command that was executed on the server. To get the exit code returned by the program, check the value of the **ExitCode** property. If an empty string is returned, this indicates that there was either no data available, or an error has occurred and the **LastError** property will return a non-zero value.

Remarks

The **Execute** method establishes a network connection with a remote server and executes the specified command. The output from the command is returned as a string. This method uses the following property values:

The value of the **UserName** and **Password** properties will be used to authenticate the session.

The value of the **Options** property will be used to specify the default options for the connection.

This method should not be used if the connection to the server must be established through a proxy server. If the connection must be made through a proxy server, then you should set the **Command** property to the specify the command to execute, call the **Connect** method to establish the connection, and then use either the **Read** or **ReadLine** methods to read the output.

When the command output is being read from the server, this method will automatically convert the data to match the end-of-line convention used on the Windows platform. This is useful when executing a command on a UNIX based system where the end-of-line is indicated by a single linefeed, while on Windows it is a carriage-return and linefeed pair. If the output contains embedded nulls or escape sequences, then this conversion will not be performed.

See Also

SshClient Class | SocketTools Namespace | SshClient.Execute Overload List

SshClient.Execute Method (String, Int32, String, String, String)

Execute a command on the server and return the output.

[Visual Basic]

```
Overloads Public Function Execute( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal password As String, _
   ByVal command As String _
) As String
```

[C#]

```
public string Execute(
   string hostName,
   int hostPort,
   string userName,
   string password,
   string command
);
```

Parameters

hostName
: A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort
: An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName
: A string which specifies the user name which will be used to authenticate the client session.

password
: A string which specifies the password which will be used to authenticate the client session.

command
: A string which specifies the command that will be executed on the server.

Return Value

A string that contains the output of the command that was executed on the server. To get the exit code returned by the program, check the value of the **ExitCode** property. If an empty string is returned, this indicates that there was either no data available, or an error has occurred and the **LastError** property will return a non-zero value.

Remarks

The **Execute** method establishes a network connection with a remote server and executes the specified command. The output from the command is returned as a string. This method uses the following property values:

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

This method should not be used if the connection to the server must be established through a proxy server. If the connection must be made through a proxy server, then you should set the **Command** property to the specify the command to execute, call the **Connect** method to establish the connection, and then use either the **Read** or **ReadLine** methods to read the output.

When the command output is being read from the server, this method will automatically convert the data to match the end-of-line convention used on the Windows platform. This is useful when executing a command on a UNIX based system where the end-of-line is indicated by a single linefeed, while on Windows it is a carriage-return and linefeed pair. If the output contains embedded nulls or escape sequences, then this conversion will not be performed.

See Also

SshClient Class | SocketTools Namespace | SshClient.Execute Overload List

SshClient.Execute Method (String, Int32, String)

Execute a command on the server and return the output.

[Visual Basic]

```
Overloads Public Function Execute( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal command As String _
) As String
```

[C#]

```
public string Execute(
   string hostName,
   int hostPort,
   string command
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

command

A string which specifies the command that will be executed on the server.

Return Value

A string that contains the output of the command that was executed on the server. To get the exit code returned by the program, check the value of the **ExitCode** property. If an empty string is returned, this indicates that there was either no data available, or an error has occurred and the **LastError** property will return a non-zero value.

Remarks

The **Execute** method establishes a network connection with a remote server and executes the specified command. The output from the command is returned as a string. This method uses the following property values:

The value of the **UserName** and **Password** properties will be used to authenticate the session.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

This method should not be used if the connection to the server must be established through a proxy server. If the connection must be made through a proxy server, then you should set the **Command** property to the specify the command to execute, call the **Connect** method to establish the connection, and then use either the **Read** or **ReadLine** methods to read the output.

When the command output is being read from the server, this method will automatically convert the data to match the end-of-line convention used on the Windows platform. This is useful when executing a command on a UNIX based system where the end-of-line is indicated by a single linefeed, while on Windows it is a carriage-return and linefeed pair. If the output contains embedded nulls or escape sequences, then this conversion will not be performed.

See Also

SshClient Class | SocketTools Namespace | SshClient.Execute Overload List

SshClient.Execute Method (String, Int32, String, String, String, Int32)

Execute a command on the server and return the output.

[Visual Basic]

```
Overloads Public Function Execute( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal password As String, _
   ByVal command As String, _
   ByVal timeout As Integer _
) As String
```

[C#]

```
public string Execute(
   string hostName,
   int hostPort,
   string userName,
   string password,
   string command,
   int timeout
);
```

Parameters

hostName
: A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort
: An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName
: A string which specifies the user name which will be used to authenticate the client session.

password
: A string which specifies the password which will be used to authenticate the client session.

command
: A string which specifies the command that will be executed on the server.

timeout
: The number of seconds that the client will wait for a response before failing the operation.

Return Value

A string that contains the output of the command that was executed on the server. To get the exit code returned by the program, check the value of the **ExitCode** property. If an empty string is returned, this indicates that there was either no data available, or an error has occurred and the **LastError** property will return a non-zero value.

Remarks

The **Execute** method establishes a network connection with a remote server and executes the specified command. The output from the command is returned as a string.

This method should not be used if the connection to the server must be established through a proxy server. If the connection must be made through a proxy server, then you should set the **Command** property to the specify the command to execute, call the **Connect** method to establish the connection, and then use either the **Read** or **ReadLine** methods to read the output.

When the command output is being read from the server, this method will automatically convert the data to match the end-of-line convention used on the Windows platform. This is useful when executing a command on a UNIX based system where the end-of-line is indicated by a single linefeed, while on Windows it is a carriage-return and linefeed pair. If the output contains embedded nulls or escape sequences, then this conversion will not be performed.

See Also

SshClient Class | SocketTools Namespace | SshClient.Execute Overload List

SshClient.Execute Method (String, Int32, String, String, String, Int32, ClientOptions)

Execute a command on the server and return the output.

[Visual Basic]

```
Overloads Public Function Execute( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal userName As String, _
   ByVal password As String, _
   ByVal command As String, _
   ByVal timeout As Integer, _
   ByVal options As ClientOptions _
) As String
```

[C#]

```
public string Execute(
   string hostName,
   int hostPort,
   string userName,
   string password,
   string command,
   int timeout,
   ClientOptions options
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

userName

A string which specifies the user name which will be used to authenticate the client session.

password

A string which specifies the password which will be used to authenticate the client session.

command

A string which specifies the command that will be executed on the server.

timeout

The number of seconds that the client will wait for a response before failing the operation.

options

One or more of the ClientOptions enumeration flags.

Return Value

A string that contains the output of the command that was executed on the server. To get the exit code returned by the program, check the value of the **ExitCode** property. If an empty string is returned, this indicates that there was either no data available, or an error has occurred and the **LastError** property will return a non-zero value.

Remarks

The **Execute** method establishes a network connection with a remote server and executes the specified command. The output from the command is returned as a string.

This method should not be used if the connection to the server must be established through a proxy server. If the connection must be made through a proxy server, then you should set the **Command** property to the specify the command to execute, call the **Connect** method to establish the connection, and then use either the **Read** or **ReadLine** methods to read the output.

When the command output is being read from the server, this method will automatically convert the data to match the end-of-line convention used on the Windows platform. This is useful when executing a command on a UNIX based system where the end-of-line is indicated by a single linefeed, while on Windows it is a carriage-return and linefeed pair. If the output contains embedded nulls or escape sequences, then this conversion will not be performed.

See Also

SshClient Class | SocketTools Namespace | SshClient.Execute Overload List

SshClient.Finalize Method

Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.

[Visual Basic]

```
Overrides Protected Sub Finalize()
```

[C#]

```
protected override void Finalize();
```

See Also

SshClient Class | SocketTools Namespace

SshClient.Initialize Method

Initialize an instance of the SshClient class.

Overload List

Initialize an instance of the SshClient class.

public bool Initialize();

Initialize an instance of the SshClient class.

public bool Initialize(string);

See Also

SshClient Class | SocketTools Namespace | Uninitialize Method

SshClient.Initialize Method ()

Initialize an instance of the SshClient class.

[Visual Basic]
```
Overloads Public Function Initialize() As Boolean
```

[C#]
```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the SshClient class, loading the networking library and allocating resources for the current thread. Typically it is not necessary to explicitly call this method because the instance of the class is initialized by the class constructor. However, if the **Uninitialize** method is called, the class must be re-initialized before any other methods are called.

See Also

SshClient Class | SocketTools Namespace | SshClient.Initialize Overload List | Uninitialize Method

SshClient.Initialize Method (String)

Initialize an instance of the SshClient class.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the SshClient class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the SshClient class.

Example

The following example shows how to use the Initialize method to initialize an instance of the class. This example assumes that the license key string has been defined in code.

```
SocketTools.SshClient sshClient = new SocketTools.SshClient();

if (sshClient.Initialize(strLicenseKey) == false)
{
    MessageBox.Show(sshClient.LastErrorString, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    return;
}
```

```
Dim sshClient As New SocketTools.SshClient

If sshClient.Initialize(strLicenseKey) = False Then
    MsgBox(sshClient.LastErrorString, vbIconExclamation)
    Exit Sub
End If
```

See Also

SshClient Class | SocketTools Namespace | SshClient.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

SshClient.Read Method

Read data from the server and store it in a byte array.

Overload List

Read data from the server and store it in a byte array.

public int Read(byte[]);

Read data from the server and store it in a byte array.

public int Read(byte[],int);

Read data from the server and store it in a string.

public int Read(ref string);

Read data from the server and store it in a string.

public int Read(ref string,int);

See Also

SshClient Class | SocketTools Namespace

SshClient.Read Method (Byte[])

Read data from the server and store it in a byte array.

[Visual Basic]

```
Overloads Public Function Read( _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Read(
   byte[] buffer
);
```

Parameters

buffer

A byte array that the data will be stored in.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the size of the byte array passed to the method. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

See Also

SshClient Class | SocketTools Namespace | SshClient.Read Overload List

SshClient.Read Method (Byte[], Int32)

Read data from the server and store it in a byte array.

[Visual Basic]

```
Overloads Public Function Read( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Read(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array that the data will be stored in.

length

An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the number of bytes specified. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

See Also

SshClient Class | SocketTools Namespace | SshClient.Read Overload List

SshClient.Read Method (String)

Read data from the server and store it in a string.

[Visual Basic]

```
Overloads Public Function Read( _
   ByRef buffer As String _
) As Integer
```

[C#]

```
public int Read(
   ref string buffer
);
```

Parameters

buffer

A string that will contain the data read from the client.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to a maximum of 4096 bytes. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

See Also

SshClient Class | SocketTools Namespace | SshClient.Read Overload List

SshClient.Read Method (String, Int32)

Read data from the server and store it in a string.

[Visual Basic]

```
Overloads Public Function Read( _
   ByRef buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Read(
   ref string buffer,
   int length
);
```

Parameters

buffer

A string that will contain the data read from the client.

length

An integer value which specifies the maximum number of bytes of data to read.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the number of bytes specified. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

See Also

SshClient Class | SocketTools Namespace | SshClient.Read Overload List

SshClient.ReadLine Method

Read up to a line of data from the server and return it in a string buffer.

Overload List

Read up to a line of data from the server and return it in a string buffer.

public bool ReadLine(ref string);

Read up to a line of data from the server and return it in a string buffer.

public bool ReadLine(ref string,int);

See Also

SshClient Class | SocketTools Namespace

SshClient.ReadLine Method (String)

Read up to a line of data from the server and return it in a string buffer.

[Visual Basic]

```
Overloads Public Function ReadLine( _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool ReadLine(
   ref string buffer
);
```

Parameters

buffer

A string which will contain the data read from the socket.

Return Value

This method returns a Boolean value which specifies if a line of data has been read. A value of **true** indicates a line of data has been read. If an error occurs or there is no more data available to read, then the method will return **false**. It is possible for data to be returned in the string buffer even if the return value is **false**. Applications should check the length of the string after the method returns to determine if any data was copied into the buffer. For example, if a timeout occurs while the method is waiting for more data to arrive on the socket, it will return zero; however, data may have already been copied into the string buffer prior to the error condition. It is the responsibility of the application to process that data, regardless of the method return value.

Remarks

The **ReadLine** method reads data from the server up to 4096 bytes in length or until an end-of-line character sequence is encountered. Unlike the **Read** method which reads arbitrary bytes of data, this method is specifically designed to return a single line of text data in a string variable. When an end-of-line character sequence is encountered, the method will stop and return the data up to that point; the string will not contain the carriage-return or linefeed characters.

There are some limitations when using the **ReadLine** method. The method should only be used to read text, never binary data. In particular, it will discard nulls, linefeed and carriage return control characters. This method will force the current thread to block until an end-of-line character sequence is processed, the read operation times out or the remote host closes its end of the socket connection. If the **Blocking** property is set to **false**, calling this method will automatically switch the client into a blocking mode, read the data and then restore the client to non-blocking mode. If another network operation is attempted while **ReadLine** is blocked waiting for data from the remote host, an error will occur. It is recommended that this method only be used with blocking connections.

The **Read** and **ReadLine** methods can be intermixed, however be aware that the **Read** method will consume any data that has already been buffered by the **ReadLine** method and this may have unexpected results.

See Also

SshClient Class | SocketTools Namespace | SshClient.ReadLine Overload List

SshClient.ReadLine Method (String, Int32)

Read up to a line of data from the server and return it in a string buffer.

[Visual Basic]

```
Overloads Public Function ReadLine( _
   ByRef buffer As String, _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool ReadLine(
   ref string buffer,
   int length
);
```

Parameters

buffer

A string which will contain the data read from the socket.

length

An integer value which specifies the maximum number of bytes of data to read.

Return Value

This method returns a Boolean value which specifies if a line of data has been read. A value of **true** indicates a line of data has been read. If an error occurs or there is no more data available to read, then the method will return **false**. It is possible for data to be returned in the string buffer even if the return value is **false**. Applications should check the length of the string after the method returns to determine if any data was copied into the buffer. For example, if a timeout occurs while the method is waiting for more data to arrive on the socket, it will return zero; however, data may have already been copied into the string buffer prior to the error condition. It is the responsibility of the application to process that data, regardless of the method return value.

Remarks

The **ReadLine** method reads data from the server up to the specified number of bytes or until an end-of-line character sequence is encountered. Unlike the **Read** method which reads arbitrary bytes of data, this method is specifically designed to return a single line of text data in a string variable. When an end-of-line character sequence is encountered, the method will stop and return the data up to that point; the string will not contain the carriage-return or linefeed characters.

There are some limitations when using the **ReadLine** method. The method should only be used to read text, never binary data. In particular, it will discard nulls, linefeed and carriage return control characters. This method will force the current thread to block until an end-of-line character sequence is processed, the read operation times out or the remote host closes its end of the socket connection. If the **Blocking** property is set to **false**, calling this method will automatically switch the client into a blocking mode, read the data and then restore the client to non-blocking mode. If another network operation is attempted while **ReadLine** is blocked waiting for data from the remote host, an error will occur. It is recommended that this method only be used with blocking connections.

The **Read** and **ReadLine** methods can be intermixed, however be aware that the **Read** method will consume any data that has already been buffered by the **ReadLine** method and this may have unexpected results.

See Also

SshClient Class | SocketTools Namespace | SshClient.ReadLine Overload List

SshClient.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. If a socket has been allocated, it will be released and any active connections will be terminated. All properties will be reset to their default values.

See Also

SshClient Class | SocketTools Namespace

SshClient.Search Method

Search for a specific character sequence in the data stream.

Overload List

Search for a specific character sequence in the data stream.

public bool Search(string);

Search for a specific character sequence in the data stream.

public bool Search(string,byte[],ref int);

Search for a specific character sequence in the data stream.

public bool Search(string,ref string);

See Also

SshClient Class | SocketTools Namespace

SshClient.Search Method (String)

Search for a specific character sequence in the data stream.

[Visual Basic]

```
Overloads Public Function Search( _
   ByVal value As String _
) As Boolean
```

[C#]

```
public bool Search(
   string value
);
```

Parameters

value

A string argument which specifies the sequence of characters to search for in the data stream. When this sequence of characters is found, the method will return.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Search** method searches for a character sequence in the data stream and stops reading when it is found. This is useful when the client wants to automate responses to the server, such as executing a command and processing the output. The method will discard any data received up to and including the specified character sequence.

See Also

SshClient Class | SocketTools Namespace | SshClient.Search Overload List

SshClient.Search Method (String, Byte[], Int32)

Search for a specific character sequence in the data stream.

[Visual Basic]
```
Overloads Public Function Search( _
   ByVal value As String, _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]
```
public bool Search(
   string value,
   byte[] buffer,
   ref int length
);
```

Parameters

value
A string argument which specifies the sequence of characters to search for in the data stream. When this sequence of characters is found, the method will return.

buffer
An byte array that will contain the output sent by the server, up to and including the search string character sequence.

length
An integer value passed by reference which should be initialized to the maximum number of bytes of data to store in the buffer. When the method returns, this value will be updated with the actual number of bytes stored in the buffer.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Search** method searches for a character sequence in the data stream and stops reading when it is found. This is useful when the client wants to automate responses to the server, such as executing a command and processing the output. The method collects the output from the server and stores it in a buffer provided by the caller. When the method returns, the buffer will contain everything sent by the server up to and including the search string.

See Also

SshClient Class | SocketTools Namespace | SshClient.Search Overload List

SshClient.Search Method (String, String)

Search for a specific character sequence in the data stream.

[Visual Basic]

```
Overloads Public Function Search( _
   ByVal value As String, _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool Search(
   string value,
   ref string buffer
);
```

Parameters

value

A string argument which specifies the sequence of characters to search for in the data stream. When this sequence of characters is found, the method will return.

buffer

An string that will contain the output sent by the server, up to and including the search string character sequence.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Search** method searches for a character sequence in the data stream and stops reading when it is found. This is useful when the client wants to automate responses to the server, such as executing a command and processing the output. The method collects the output from the server and stores it in a buffer provided by the caller. When the method returns, the buffer will contain everything sent by the server up to and including the search string.

See Also

SshClient Class | SocketTools Namespace | SshClient.Search Overload List

SshClient.Uninitialize Method

Uninitialize the class library and release any resources allocated for the current thread.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method terminates any active connection, releases resources allocated for the current thread and unloads the networking library. After this method has been called, no further client operations may be performed until the class instance has been re-initialized.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the class is no longer needed.

See Also

SshClient Class | SocketTools Namespace | Initialize Method

SshClient.Write Method

Write one or more bytes of data to the server.

Overload List

Write one or more bytes of data to the server.

public int Write(byte[]);

Write one or more bytes of data to the server.

public int Write(byte[],int);

Write a character to the server.

public int Write(char);

Write one or more characters to the server.

public int Write(char,int);

Write a string of characters to the server.

public int Write(string);

Write a string of characters to the server.

public int Write(string,int);

See Also

SshClient Class | SocketTools Namespace

SshClient.Write Method (Byte[])

Write one or more bytes of data to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Write(
   byte[] buffer
);
```

Parameters

buffer

A byte array that contains the data to be written to the server.

Return Value

An integer value which specifies the number of bytes actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

SshClient Class | SocketTools Namespace | SshClient.Write Overload List

SshClient.Write Method (Byte[], Int32)

Write one or more bytes of data to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Write(
   byte[] buffer,
   int length
);
```

Parameters

buffer
A byte array that contains the data to be written to the server.

length
An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

SshClient Class | SocketTools Namespace | SshClient.Write Overload List

SshClient.Write Method (Char)

Write a character to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal key As Char _
) As Integer
```

[C#]

```
public int Write(
   char key
);
```

Parameters

key

A character which will be written to the server.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one character to the server. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

SshClient Class | SocketTools Namespace | SshClient.Write Overload List

SshClient.Write Method (Char, Int32)

Write one or more characters to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal key As Char, _
   ByVal repeat As Integer _
) As Integer
```

[C#]

```
public int Write(
   char key,
   int repeat
);
```

Parameters

key

A character which will be written to the server.

repeat

The number of characters that will be written to the server.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more characters to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

SshClient Class | SocketTools Namespace | SshClient.Write Overload List

SshClient.Write Method (String)

Write a string of characters to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String _
) As Integer
```

[C#]

```
public int Write(
   string buffer
);
```

Parameters

buffer

A string which contains the data to be written to the server.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

The string will be converted to an array of bytes before being written to the socket. By default, the character encoding used will be for the current locale. Depending on the contents of the string, the number of bytes written may be different than the string length specified. This is because the conversion from Unicode to a byte array may result in a multi-byte character sequence.

See Also

SshClient Class | SocketTools Namespace | SshClient.Write Overload List

SshClient.Write Method (String, Int32)

Write a string of characters to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Write(
   string buffer,
   int length
);
```

Parameters

buffer

A string which contains the data to be written to the server.

length

An integer value which specifies the maximum number of characters to write. This value cannot be larger than the length of the string specified by the caller.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

The string will be converted to an array of bytes before being written to the socket. By default, the character encoding used will be for the current locale. Depending on the contents of the string, the number of bytes written may be different than the string length specified. This is because the conversion from Unicode to a byte array may result in a multi-byte character sequence.

See Also

SshClient Class | SocketTools Namespace | SshClient.Write Overload List

SshClient.WriteLine Method

Send an empty line of text to the server, terminated by a carriage-return and linefeed.

Overload List

Send an empty line of text to the server, terminated by a carriage-return and linefeed.

```
public bool WriteLine();
```

Send a line of text to the server, terminated by a carriage-return and linefeed.

```
public bool WriteLine(string);
```

Send a line of text to the server, terminated by a carriage-return and linefeed.

```
public bool WriteLine(string,ref int);
```

See Also

SshClient Class | SocketTools Namespace

SshClient.WriteLine Method ()

Send an empty line of text to the server, terminated by a carriage-return and linefeed.

[Visual Basic]

```
Overloads Public Function WriteLine() As Boolean
```

[C#]

```
public bool WriteLine();
```

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

The **WriteLine** method will send an empty line of text, terminated by a carriage-return and linefeed. Calling this method will force the application to block until the complete line of text has been written, the write operation times out or the remote host aborts the connection. If this method is called with the **Blocking** property set to **false**, it will automatically switch the client into a blocking mode, send the data and then restore the client to non-blocking mode. If another network operation is attempted while the **WriteLine** method is blocked sending data to the remote host, an error will occur. It is recommended that this method only be used with blocking connections.

The **Write** and **WriteLine** methods can be safely intermixed.

See Also

SshClient Class | SocketTools Namespace | SshClient.WriteLine Overload List

SshClient.WriteLine Method (String)

Send a line of text to the server, terminated by a carriage-return and linefeed.

[Visual Basic]

```
Overloads Public Function WriteLine( _
   ByVal buffer As String _
) As Boolean
```

[C#]

```
public bool WriteLine(
   string buffer
);
```

Parameters

buffer

A string which contains the data that will be sent to the remote host. The data will always be terminated with a carriage-return and linefeed control character sequence. If the string is empty, then a only a carriage-return and linefeed are written to the socket. Note that if the string contains a null character, any data that follows the null character will be discarded.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

The **WriteLine** method should only be used to send text, never binary data. In particular, this method will discard any data that follows a null character and will append linefeed and carriage return control characters to the data stream. Calling this method will force the current thread to block until the complete line of text has been written, the write operation times out or the remote host aborts the connection. If this method is called with the **Blocking** property set to **false**, it will automatically switch the client into a blocking mode, send the data and then restore the client to non-blocking mode. If another network operation is attempted while the **WriteLine** method is blocked sending data to the remote host, an error will occur. It is recommended that this method only be used with blocking connections.

The string will be converted to an array of bytes before being written to the socket. By default, the character encoding used will be for the current locale. Depending on the contents of the string, the number of bytes written may be different than the string length specified. This is because the conversion from Unicode to a byte array may result in a multi-byte character sequence.

The **Write** and **WriteLine** methods can be safely intermixed.

See Also

SshClient Class | SocketTools Namespace | SshClient.WriteLine Overload List

SshClient.WriteLine Method (String, Int32)

Send a line of text to the server, terminated by a carriage-return and linefeed.

[Visual Basic]

```
Overloads Public Function WriteLine( _
   ByVal buffer As String, _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool WriteLine(
   string buffer,
   ref int length
);
```

Parameters

buffer

A string which contains the data that will be sent to the remote host. The data will always be terminated with a carriage-return and linefeed control character sequence. If the string is empty, then a only a carriage-return and linefeed are written to the socket. Note that if the string contains a null character, any data that follows the null character will be discarded.

length

An integer value which specifies the maximum number of characters to write. This value cannot be larger than the length of the string specified by the caller.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

The **WriteLine** method should only be used to send text, never binary data. In particular, this method will discard any data that follows a null character and will append linefeed and carriage return control characters to the data stream. Calling this method will force the current thread to block until the complete line of text has been written, the write operation times out or the remote host aborts the connection. If this method is called with the **Blocking** property set to **false**, it will automatically switch the client into a blocking mode, send the data and then restore the client to non-blocking mode. If another network operation is attempted while the **WriteLine** method is blocked sending data to the remote host, an error will occur. It is recommended that this method only be used with blocking connections.

The string will be converted to an array of bytes before being written to the socket. By default, the character encoding used will be for the current locale. Depending on the contents of the string, the number of bytes written may be different than the string length specified. This is because the conversion from Unicode to a byte array may result in a multi-byte character sequence.

The **Write** and **WriteLine** methods can be safely intermixed.

See Also

SshClient Class | SocketTools Namespace | SshClient.WriteLine Overload List

SshClient Events

The events of the **SshClient** class are listed below. For a complete list of **SshClient** class members, see the SshClient Members topic.

Public Instance Events

⚡ OnCancel	Occurs when a blocking client operation is canceled.
⚡ OnConnect	Occurs when a connection is established with the remote host.
⚡ OnDisconnect	Occurs when the remote host disconnects from the local system.
⚡ OnError	Occurs when an client operation fails.
⚡ OnRead	Occurs when data is available to be read from the client.
⚡ OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
⚡ OnWrite	Occurs when data can be written to the client.

See Also

SshClient Class | SocketTools Namespace

SshClient.OnCancel Event

Occurs when a blocking client operation is canceled.

[Visual Basic]

```
Public Event OnCancel As EventHandler
```

[C#]

```
public event EventHandler OnCancel;
```

Remarks

The **OnCancel** event is generated when a blocking client operation, such as sending or receiving data, is canceled with the **Cancel** method. To assist in determining which operation was canceled, check the value of the **Status** property.

See Also

SshClient Class | SocketTools Namespace

SshClient.OnConnect Event

Occurs when a connection is established with the remote host.

[Visual Basic]
```
Public Event OnConnect As EventHandler
```

[C#]
```
public event EventHandler OnConnect;
```

Remarks

The **OnConnect** event occurs when a connection is made with a remote host as a result of a **Connect** method call. When the **Connect** method is called and the **Blocking** property is set to **false**, a socket is created but the connection is not actually established until after this event occurs. Between the time connection process is started and this event fires, no operation may be performed on the client other than calling the **Disconnect** method.

This event is only generated if the client is in non-blocking mode.

See Also

SshClient Class | SocketTools Namespace

SshClient.OnDisconnect Event

Occurs when the remote host disconnects from the local system.

[Visual Basic]
```
Public Event OnDisconnect As EventHandler
```

[C#]
```
public event EventHandler OnDisconnect;
```

Remarks

The **OnDisconnect** event occurs when the remote host closes its connection, terminating the client session with the application. Because there may still be data in the client receive buffers, you should continue to read data from the client until the **Read** method returns a value of 0. Once all of the data has been read, you should call the **Disconnect** method to close the local socket and release the resources allocated for the client.

This event is only generated if the client is in non-blocking mode.

See Also

SshClient Class | SocketTools Namespace

SshClient.OnError Event

Occurs when an client operation fails.

[Visual Basic]

```
Public Event OnError As OnErrorEventHandler
```

[C#]

```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type SshClient.ErrorEventArgs containing data related to this event. The following **SshClient.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a client operation fails.

See Also

SshClient Class | SocketTools Namespace

SshClient.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see SshClient.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.SshClient.ErrorEventArgs

[Visual Basic]

```
Public Class SshClient.ErrorEventArgs
    Inherits EventArgs
```

[C#]

```
public class SshClient.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SshClient (in SocketTools.SshClient.dll)

See Also

SshClient.ErrorEventArgs Members | SocketTools Namespace

SshClient.ErrorEventArgs Members

SshClient.ErrorEventArgs overview

Public Instance Constructors

SshClient.ErrorEventArgs Constructor	Initializes a new instance of the SshClient.ErrorEventArgs class.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

SshClient.ErrorEventArgs Class | SocketTools Namespace

SshClient.ErrorEventArgs Constructor

Initializes a new instance of the SshClient.ErrorEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public SshClient.ErrorEventArgs();
```

See Also

SshClient.ErrorEventArgs Class | SocketTools Namespace

SshClient.ErrorEventArgs Properties

The properties of the **SshClient.ErrorEventArgs** class are listed below. For a complete list of **SshClient.ErrorEventArgs** class members, see the SshClient.ErrorEventArgs Members topic.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

SshClient.ErrorEventArgs Class | SocketTools Namespace

SshClient.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]
```
Public ReadOnly Property Description As String
```

[C#]
```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

SshClient.ErrorEventArgs Class | SocketTools Namespace | Error Property

SshClient.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public SshClient.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

SshClient.ErrorEventArgs Class | SocketTools Namespace | Description Property

SshClient.OnRead Event

Occurs when data is available to be read from the client.

[Visual Basic]

```
Public Event OnRead As EventHandler
```

[C#]

```
public event EventHandler OnRead;
```

Remarks

The **OnRead** event occurs when data is available to be read from the client. This event is level-triggered, which means that once this event fires, it will not occur again until some data has been read from the client. This design prevents an application from being flooded with event notifications. It is recommended that your application read all of the available data from the server and store it in a local buffer for processing. See the example below.

This event is only generated if the client is in non-blocking mode.

Example

```
Private Sub Socket_OnRead(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Socket.OnRead
    Dim strBuffer As String
    Dim nRead As Integer

    Do
        ' Read up to m_nBufferSize bytes of data from the server
        nRead = Socket.Read(strBuffer, m_nBufferSize)

        If nRead > 0 Then
            ' Append the data to an internal buffer for processing
            m_dataBuffer = m_dataBuffer + strBuffer
        End If
    Loop Until nRead < 1

    ProcessData()
End Sub
```

See Also

SshClient Class | SocketTools Namespace

SshClient.OnTimeout Event

Occurs when a blocking operation fails to complete before the timeout period elapses.

[Visual Basic]

```
Public Event OnTimeout As EventHandler
```

[C#]

```
public event EventHandler OnTimeout;
```

Remarks

The **OnTimeout** event occurs when a blocking operation, such as sending or receiving data on the client, fails to complete before the specified timeout period elapses. The timeout period for a blocking operation can be adjusted by setting the **Timeout** property.

This event is only generated if the client is in blocking mode.

See Also

SshClient Class | SocketTools Namespace

SshClient.OnWrite Event

Occurs when data can be written to the client.

[Visual Basic]

```
Public Event OnWrite As EventHandler
```

[C#]

```
public event EventHandler OnWrite;
```

Remarks

The **OnWrite** event occurs when the application can write data to the client. This event will typically occur when a connection is first established with the remote host, and after the **Write** method has failed because there was insufficient memory available in the client send buffers. In the second case, when some of the buffered data has been successfully sent to the remote host and there is space available in the send buffers, this event is used to signal the application that it may attempt to send more data.

This event is only generated if the client is in non-blocking mode.

See Also

SshClient Class | SocketTools Namespace

SshClient.ClientOptions Enumeration

Specifies the options that the SshClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum SshClient.ClientOptions
```

```
[C#]
[Flags]
public enum SshClient.ClientOptions
```

Remarks

The SshClient class uses the **ClientOptions** enumeration to specify one or more options to be used when establishing a connection with a remote host. Multiple options may be specified if necessary.

Members

Member Name	Description	Value
optionNone	No option specified.	0
optionDefault	The default connection option. This is the same as specifying **optionNone**.	0
optionKeepAlive	This option specifies the library should attempt to maintain an idle client session for long periods of time. This option is only necessary if you expect that the connection will be held open for more than two hours. This option is the same as setting the **KeepAlive** property to a value of true.	1
optionNoPTY	This option specifies that a pseudoterminal (PTY) should not be created for the client session. This option is automatically set if the **Command** property specifies a command to be executed on the server.	2
optionNoShell	This option specifies that a command shell should not be used when executing a command on the remote host.	4
optionNoAuthRSA	This option specifies that RSA authentication should not be used with SSH-1 connections. This option is ignored with SSH-2 connections and should only be specified if required by the remote host.	8
optionNoPwdNul	This option specifies the user password	16

	cannot be terminated with a null byte. This option is ignored with SSH-2 connections and should only be specified if required by the remote host.	
optionNoRekey	This option specifies the client should never attempt a repeat key exchange with the server. Some SSH servers do not support rekeying the session, and this can cause the client to become non-responsive or abort the connection after being connected for an hour.	32
optionCompatSID	This compatibility option changes how the session ID is handled during public key authentication with older SSH servers. This option should only be specified when connecting to servers that use OpenSSH 2.2.0 or earlier versions.	64
optionCompatHMAC	This compatibility option changes how the HMAC authentication codes are generated. This option should only be specified when connecting to servers that use OpenSSH 2.2.0 or earlier versions.	128
optionFreeThread	This option specifies that class methods may be called from any thread, and not only the thread that established the connection. Using this option disables certain internal safety checks that are made by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads.	524288

Requirements

Namespace: SocketTools

Assembly: SocketTools.SshClient (in SocketTools.SshClient.dll)

See Also

SocketTools Namespace

SshClient.ClientStatus Enumeration

Specifies the status values that may be returned by the SshClient class.

[Visual Basic]
```
Public Enum SshClient.ClientStatus
```

[C#]
```
public enum SshClient.ClientStatus
```

Remarks

The SshClient class uses the **ClientStatus** enumeration to identify the current status of the client.

Members

Member Name	Description
statusUnused	A client session has not been created. Attempts to perform any network operations, such as sending or receiving data, will generate an error.
statusIdle	A client session has been created, but is not currently in use. A blocking socket operation can be executed at this point.
statusConnect	The client is in the process of establishing a connection with a remote host.
statusAuthenticate	The client is authenticating the session.
statusRead	The client is in the process of receiving data from a remote host.
statusWrite	The client is in the process of sending data to a remote host.
statusDisconnect	The client session is being closed and subsequent attempts to access the client will result in an error.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SshClient (in SocketTools.SshClient.dll)

See Also

SocketTools Namespace

SshClient.ControlCodes Enumeration

Specifies the control codes that the SshClient class may send to the server.

[Visual Basic]

```
Public Enum SshClient.ControlCodes
```

[C#]

```
public enum SshClient.ControlCodes
```

Remarks

The SshClient class uses the **ControlCodes** enumeration to specify the control code that should be sent to the server using the **Control** method. This enables an application to send control messages to the server, which can cause it to take specific actions such as simulate a terminal break or request that the key exchange be performed again. Note that some control messages are not supported by the SSH 1.0 protocol, in which case the control message is ignored.

Members

Member Name	Description
controlBreak	Sends a control message to the remote host which simulates a break signal on a physical terminal. This is used by some operating systems as an instruction to enter a privileged configuration mode. Note that this is not the same as sending an interrupt character such as Ctrl+C to the remote host. This control code is ignored for SSH 1.0 sessions. This is the same as calling the **Break** method.
controlNoop	Sends a control message to the remote host, but it does not perform any operation. This is typically used by clients to prevent the server from automatically closing a session that has been idle for a long period of time.
controlEof	Sends a control message to the remote host indicating that the client has finished sending data. Note that this option is normally not used with interactive terminal sessions, and should only be used when required by the server.
controlPing	Sends a control message to the remote host which is used to test whether or not the remote host is responsive to the client. This is typically used by clients to attempt to detect if the connection to the remote host is still active.
controlRekey	Sends a control message to the remote host requesting that the key exchange be performed again. This control code is ignored for SSH 1.0 sessions.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SshClient (in SocketTools.SshClient.dll)

See Also

SocketTools Namespace

SshClient.ErrorCode Enumeration

Specifies the error codes returned by the SshClient class.

[Visual Basic]

```
Public Enum SshClient.ErrorCode
```

[C#]

```
public enum SshClient.ErrorCode
```

Remarks

The SshClient class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.
errorProxyAuthenticationRequired	Proxy authentication required.

errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.
errorUnsupportedMediaType	Request specified an unsupported media type.

errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.
errorNoMoreMailboxes	No more mailboxes exist on this server.

errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.
errorInvalidThreadHandle	The specified thread handle is no longer valid.

errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been defined.

errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or unrecognized.

errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SshClient (in SocketTools.SshClient.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

SshClient.LineMode Enumeration

Specifies how end-of-line character sequences are sent to the server.

[Visual Basic]
```
Public Enum SshClient.LineMode
```

[C#]
```
public enum SshClient.LineMode
```

Members

Member Name	Description
newLineDefault	There are no changes to how data is sent to the server. Any carriage return or linefeed characters that are sent using the **Write** method will be sent as-is. The **WriteLine** method will terminate each line of text with a carriage return and linefeed (CRLF) sequence. This is the default line mode that is set when a new connection is established.
newLineCR	A carriage return is used as the end-of-line character. Any data sent using the **Write** method that contains only a linefeed (LF) character or a carriage return and linefeed (CRLF) sequence to indicate the end-of-line will be replaced by a carriage return (CR) character. The **WriteLine** method will terminate each line of text with a single carriage return character.
newLineLF	A linefeed is used as the end-of-line character. Any data sent using the **Write** method that contains only a carriage return (CR) character or a carriage return an linefeed (CRLF) sequence to indicate the end-of-line will be replaced by a linefeed (LF) character. The **WriteLine** method will terminate each line of text with a single linefeed character.
newLineCRLF	A carriage return and linefeed (CRLF) character sequence is used to indicate the end-of-line. Any data sent using the Write method that contains only a carriage return (CR) or linefeed (LF) will be replaced by a carriage return and linefeed. The **WriteLine** method will terminate each line of text with a carriage return and linefeed sequence.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SshClient (in SocketTools.SshClient.dll)

See Also

SocketTools Namespace | NewLine Property (SocketTools.SshClient)

SshClient.ProxyTypes Enumeration

Specifies the type of proxy servers that the SshClient class supports.

[Visual Basic]

```
Public Enum SshClient.ProxyTypes
```

[C#]

```
public enum SshClient.ProxyTypes
```

Remarks

The SshClient class uses the **ProxyTypes** enumeration to specify the type of proxy server that the connection should be established through, if any. By default, no proxy server is used when establishing a connection.

Members

Member Name	Description
proxyNone	No proxy server. A direct connection will be established with the server.
proxyHttp	Establish a connection through a proxy server using the Hypertext Transfer Protocol.
proxyTelnet	Establish a connection through a proxy server using the Telnet protocol.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SshClient (in SocketTools.SshClient.dll)

See Also

SocketTools Namespace

SshClient.SecureCipherAlgorithm Enumeration

Specifies the encryption algorithms that the SshClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum SshClient.SecureCipherAlgorithm
```

[C#]

```
[Flags]
public enum SshClient.SecureCipherAlgorithm
```

Remarks

The SshClient class uses the **SecureCipherAlgorithm** enumeration to identify which encryption algorithm was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
cipherNone	No cipher has been selected. A secure connection has not been established with the remote host.	0
cipherRC2	The RC2 block cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.	1
cipherRC4	The RC4 stream cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.	2
cipherRC5	The RC5 block cipher was selected. This is a variable key length cipher which supports keys up to 2040 bits, in 8-bit increments.	4
cipherDES	The DES (Data Encryption Standard) block cipher was selected. This is a fixed key length cipher using 56-bit keys.	8
cipherDES3	The Triple DES block cipher was selected. This cipher encrypts the data three times using different keys, effectively using a 168-bit key length.	16
cipherDESX	A variant of the DES block cipher which XORs an extra 64-bits of the key before and after the plaintext has been encrypted, increasing the key size to 184 bits.	32
cipherAES	The Advanced Encryption Standard	64

	cipher (also known as the Rijndael cipher) is a fixed block size cipher which use a key size of 128, 192 or 256 bits. This cipher is supported on Windows XP SP3 SP3 and later versions of the operating system.	
cipherSkipjack	The Skipjack block cipher was selected. This is a fixed key length cipher, using 80-bit keys.	128
cipherBlowfish	The Blowfish block cipher was selected. This is a variable key length cipher up to 448 bits, using a 64-bit block size.	256

Requirements

Namespace: SocketTools

Assembly: SocketTools.SshClient (in SocketTools.SshClient.dll)

See Also

SocketTools Namespace

SshClient.SecureHashAlgorithm Enumeration

Specifies the hash algorithms that the SshClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum SshClient.SecureHashAlgorithm
```

[C#]

```
[Flags]
public enum SshClient.SecureHashAlgorithm
```

Remarks

The SshClient class uses the **SecureHashAlgorithm** enumeration to identify the message digest (hash) algorithm that was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
hashNone	No hash algorithm has been selected. This is not a secure connection with the server.	0
hashMD5	The MD5 algorithm was selected. This algorithm produces a 128-bit message digest. This algorithm is no longer considered to be cryptographically secure.	1
hashSHA	The SHA-1 algorithm was selected. This algorithm produces a 160-bit message digest. This algorithm is no longer considered to be cryptographically secure.	2
hashSHA256	The SHA-256 algorithm was selected. This algorithm produces a 256-bit message digest.	4

Requirements

Namespace: SocketTools

Assembly: SocketTools.SshClient (in SocketTools.SshClient.dll)

See Also

SocketTools Namespace

SshClient.SecureKeyAlgorithm Enumeration

Specifies the key exchange algorithms that the SshClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum SshClient.SecureKeyAlgorithm
```

[C#]
```
[Flags]
public enum SshClient.SecureKeyAlgorithm
```

Remarks

The SshClient class uses the **SecureKeyAlgorithm** enumeration to identify the key exchange algorithm that was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
keyExchangeNone	No key exchange algorithm has been selected. This is not a secure connection with the server.	0
keyExchangeRSA	The RSA public key exchange algorithm has been selected.	1
keyExchangeKEA	The KEA public key exchange algorithm has been selected. This is an improved version of the Diffie-Hellman public key algorithm.	2
keyExchangeDH	The Diffie-Hellman public key exchange algorithm has been selected.	4
keyExchangeECDH	The Elliptic Curve Diffie-Hellman key exchange algorithm was selected. This is a variant of the Diffie-Hellman algorithm which uses elliptic curve cryptography. This key exchange algorithm is only supported on Windows XP SP3 SP3 and later versions of the operating system.	8

Requirements

Namespace: SocketTools

Assembly: SocketTools.SshClient (in SocketTools.SshClient.dll)

See Also

SocketTools Namespace

SshClient.SecurityProtocols Enumeration

Specifies the security protocols that the SshClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum SshClient.SecurityProtocols
```

[C#]
```
[Flags]
public enum SshClient.SecurityProtocols
```

Remarks

The SshClient class uses the **SecurityProtocols** enumeration to specify one or more security protocols to be used when establishing a connection with a remote host. Multiple protocols may be specified if necessary and the actual protocol used will be negotiated with the remote host. It is recommended that most applications use **protocolDefault** when creating a secure connection.

Members

Member Name	Description	Value
protocolNone	No security protocol has been selected. Because all connections to an SSH server are secure, this value indicates that a connection has not been established.	0
protocolSSH1	The Secure Shell 1.0 protocol has been selected. This version of the protocol has been deprecated and is no longer widely used. It is not recommended that this version of the protocol be used to establish a connection.	256
protocolSSH2	The Secure Shell 2.0 protocol has been selected. This is the most commonly used version of the protocol. It is recommended that this version of the protocol be used unless the server explicitly requires the client to use an earlier version.	512
protocolDefault	Either version 1.0 or 2.0 of the protocol should be used when establishing the connection. The correct version of the protocol will be automatically selected, based on which version is supported by the server.	512
protocolUnknown	An unknown or unsupported security protocol has been specified. This value indicates an error condition.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.SshClient (in SocketTools.SshClient.dll)

See Also

SocketTools Namespace

SshClient.TraceOptions Enumeration

Specifies the logging options that the SshClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum SshClient.TraceOptions
```

[C#]

```
[Flags]
public enum SshClient.TraceOptions
```

Remarks

The SshClient class uses the **TraceOptions** enumeration to specify what kind of debugging information is written to the trace logfile. These options are only meaningful when trace logging is enabled by setting the **Trace** property to **true**.

Members

Member Name	Description	Value
traceDefault	The default trace logging option. This is the same as specifying the **traceInfo** option.	0
traceInfo	All network function calls are written to the trace file. This is the default value.	0
traceError	Only those network function calls which fail are recorded in the trace file.	1
traceWarning	Only those network function calls which fail, or return values which indicate a warning, are recorded in the trace file.	2
traceHexDump	All network function calls are written to the trace file, plus all the data that is sent or received is displayed, in both ASCII and hexadecimal format.	4
traceProcess	All function calls in the current process are logged, rather than only those functions in the current thread. This option is useful for multithreaded applications that are using worker threads.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.SshClient (in SocketTools.SshClient.dll)

See Also

SocketTools Namespace

SshClient.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub SshClient.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void SshClient.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ErrorEventArgs that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SshClient (in SocketTools.SshClient.dll)

See Also

SocketTools Namespace

SshClient.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see SshClient.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.SshClient.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class SshClient.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class SshClient.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the SshClient class.

Example

```
<Assembly: SocketTools.SshClient.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.SshClient.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.SshClient (in SocketTools.SshClient.dll)

See Also

SshClient.RuntimeLicenseAttribute Members | SocketTools Namespace

SshClient.RuntimeLicenseAttribute Members

SshClient.RuntimeLicenseAttribute overview

Public Instance Constructors

SshClient.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

SshClient.RuntimeLicenseAttribute Class | SocketTools Namespace

SshClient.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public SshClient.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the SshClient class.

See Also

SshClient.RuntimeLicenseAttribute Class | SocketTools Namespace

SshClient.RuntimeLicenseAttribute Properties

The properties of the **SshClient.RuntimeLicenseAttribute** class are listed below. For a complete list of **SshClient.RuntimeLicenseAttribute** class members, see the SshClient.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

SshClient.RuntimeLicenseAttribute Class | SocketTools Namespace

SshClient.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

SshClient.RuntimeLicenseAttribute Class | SocketTools Namespace

SshClientException Class

The exception that is thrown when a client error occurs.

For a list of all members of this type, see SshClientException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.SshClientException

[Visual Basic]
```
Public Class SshClientException
    Inherits ApplicationException
```

[C#]
```
public class SshClientException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A SshClientException is thrown by the SshClient class when an error occurs.

The default constructor for the SshClientException class sets the **ErrorCode** property to the last client error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.SshClient (in SocketTools.SshClient.dll)

See Also

SshClientException Members | SocketTools Namespace

SshClientException Members

SshClientException overview

Public Instance Constructors

SshClientException	Overloaded. Initializes a new instance of the SshClientException class.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

SshClientException Class | SocketTools Namespace

SshClientException Constructor

Initializes a new instance of the SshClientException class with the last network error code.

Overload List

Initializes a new instance of the SshClientException class with the last network error code.

public SshClientException();

Initializes a new instance of the SshClientException class with a specified error number.

public SshClientException(int);

Initializes a new instance of the SshClientException class with a specified error message.

public SshClientException(string);

Initializes a new instance of the SshClientException class with a specified error message and a reference to the inner exception that is the cause of this exception.

public SshClientException(string,Exception);

See Also

SshClientException Class | SocketTools Namespace

SshClientException Constructor ()

Initializes a new instance of the SshClientException class with the last network error code.

[Visual Basic]

```
Overloads Public Sub New()
```

[C#]

```
public SshClientException();
```

Remarks

The ctor constructor sets the **ErrorCode** property to the last client error that occurred. For more information about the errors that may occur, refer to the SshClient.ErrorCode enumeration.

See Also

SshClientException Class | SocketTools Namespace | SshClientException Constructor Overload List

SshClientException Constructor (String)

Initializes a new instance of the SshClientException class with a specified error message.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]

```
public SshClientException(
   string message
);
```

Parameters

message

The error message that explains the reason for the exception.

Remarks

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

SshClientException Class | SocketTools Namespace | SshClientException Constructor Overload List

SshClientException Constructor (String, Exception)

Initializes a new instance of the SshClientException class with a specified error message and a reference to the inner exception that is the cause of this exception.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal innerException As Exception _
)
```

[C#]

```
public SshClientException(
   string message,
   Exception innerException
);
```

Parameters

message

The error message that explains the reason for the exception.

innerException

The exception that is the cause of the current exception. If the ***innerException*** parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

SshClientException Class | SocketTools Namespace | SshClientException Constructor Overload List

SshClientException Constructor (Int32)

Initializes a new instance of the SshClientException class with a specified error number.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

[C#]

```
public SshClientException(
   int code
);
```

Parameters

code
An integer value which specifies an error code.

Remarks

This constructor sets the **ErrorCode** property to the specified error code. For more information about the errors that may occur, refer to the SshClient.ErrorCode enumeration.

See Also

SshClientException Class | SocketTools Namespace | SshClientException Constructor Overload List

SshClientException Properties

The properties of the **SshClientException** class are listed below. For a complete list of **SshClientException** class members, see the SshClientException Members topic.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

SshClientException Class | SocketTools Namespace

SshClientException.ErrorCode Property

Gets a value which specifies the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property ErrorCode As ErrorCode
```

[C#]

```
public SshClient.ErrorCode ErrorCode {get;}
```

Property Value

Returns a SshClient.ErrorCode enumeration value which specifies the error code.

Remarks

The **ErrorCode** property returns the error code that specifies the cause of the exception.

The default constructor for the SshClientException class sets the error code to the last network error that occurred. For more information about the errors that may occur, refer to the SshClient.ErrorCode enumeration.

See Also

SshClientException Class | SocketTools Namespace

SshClientException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]

```
Overrides Public ReadOnly Property Message As String
```

[C#]

```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

SshClientException Class | SocketTools Namespace

SshClientException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception. This value is the same as the values returned by the Windows Sockets API. For more information about socket error codes, see the Windows Socket Version 2 API error code documentation in MSDN.

See Also

SshClientException Class | SocketTools Namespace

SshClientException Methods

The methods of the **SshClientException** class are listed below. For a complete list of **SshClientException** class members, see the SshClientException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

SshClientException Class | SocketTools Namespace

SshClientException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]

```
Overrides Public Function ToString() As String
```

[C#]

```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

SshClientException Class | SocketTools Namespace

TelnetClient Class

Implements the Telnet protocol.

For a list of all members of this type, see TelnetClient Members.

System.Object
  SocketTools.TelnetClient

[Visual Basic]

```
Public Class TelnetClient
    Implements IDisposable
```

[C#]

```
public class TelnetClient : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The Telnet protocol is used to establish a connection with a server which provides a virtual terminal session for a user. Its functionality is similar to how character based consoles and serial terminals work, enabling a user to login to the server, execute commands and interact with applications running on the remote host. The TelnetClient class provides an interface for establishing the connection, negotiating certain options (such as whether characters will be echoed back to the client) and handling the standard I/O functions needed by the program. The class also provides functions that enable a program to easily scan the data stream for specific sequences of characters, making it very simple to write light-weight client interfaces to applications running on the server. This class can be combined with the **SocketTools.Terminal** component to provide complete terminal emulation services for a standard ANSI or DEC-VT220 terminal.

Requirements

Namespace: SocketTools

Assembly: SocketTools.TelnetClient (in SocketTools.TelnetClient.dll)

See Also

TelnetClient Members | SocketTools Namespace

TelnetClient Members

TelnetClient overview

Public Static (Shared) Fields

telnetPortDefault	A constant value which specifies the default port number.
telnetPortSecure	A constant value which specifies the default port number for a secure connection.
telnetTimeout	A constant value which specifies the default timeout period.

Public Instance Constructors

TelnetClient Constructor	Initializes a new instance of the TelnetClient class.

Public Instance Properties

AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
Blocking	Gets and sets a value which indicates if the client is in blocking mode.
CertificateExpires	Get a value that specifies the date that the security certificate expires.
CertificateIssued	Get a value that specifies the date that the security certificate was issued.
CertificateIssuer	Get a value that provides information about the organization that issued the certificate.
CertificateName	Gets and sets a value that specifies the name of the client certificate.
CertificatePassword	Gets and sets the password associated with the client certificate.
CertificateStatus	Gets a value which indicates the status of the security certificate returned by the remote host.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateSubject	Gets a value that provides information about the organization that the server certificate was issued to.
CertificateUser	Gets and sets the user that owns the client certificate.
CipherStrength	Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.
CodePage	Gets and sets the code page used when reading

	and writing text.
Handle	Gets a value that specifies the client handle allocated for the current session.
HashStrength	Gets a value which specifies the length of the message digest that was selected for a secure connection.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsConnected	Gets a value which indicates if a connection to the remote host has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsReadable	Gets a value which indicates if there is data available to be read from the socket connection to the server.
IsThere	Gets a value which indicates if the the remote host is responsive
IsWritable	Gets a value which indicates if data can be written to the client without blocking.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets the local Internet address that the client is bound to.
LocalEcho	Enables and disables the echoing of characters by the remote host.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
Options	Gets and sets a value which specifies one or more client options.
Password	Gets and sets the password used to authenticate the client session.
RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.

Secure	Gets and sets a value which specifies if a secure connection is established.
SecureCipher	Gets a value that specifies the encryption algorithm used for a secure connection.
SecureHash	Gets a value that specifies the message digest algorithm used for a secure connection.
SecureKeyExchange	Gets a value that specifies the key exchange algorithm used for a secure connection.
SecureProtocol	Gets and sets a value which specifies the protocol used for a secure connection.
Status	Gets a value which specifies the current status of the client.
Terminal	Gets and sets the terminal type used for a remote login session.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
UserName	Gets and sets the username used to authenticate the client session.
Version	Gets a value which returns the current version of the TelnetClient class library.

Public Instance Methods

Abort	Aborts the current session and terminates the connection.
AttachThread	Attach an instance of the class to the current thread
Break	Sends a break signal to the remote host.
Cancel	Cancel the current blocking client operation.
Connect	Overloaded. Establish a connection with a remote host.
Disconnect	Terminate the connection with a remote host.

Dispose	Overloaded. Releases all resources used by TelnetClient.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the TelnetClient class.
Login	Overloaded. Login to the remote server.
Read	Overloaded. Read data from the server and store it in a byte array.
ReadLine	Overloaded. Read up to a line of data from the server and return it in a string buffer.
Reset	Reset the internal state of the object, resetting all properties to their default values.
Search	Overloaded. Search for a specific character sequence in the data stream.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
Write	Overloaded. Write one or more bytes of data to the server.
WriteLine	Overloaded. Send a line of text to the server, terminated by a carriage-return and linefeed.

Public Instance Events

OnCancel	Occurs when a blocking client operation is canceled.
OnConnect	Occurs when a connection is established with the remote host.
OnDisconnect	Occurs when the remote host disconnects from the local system.
OnError	Occurs when an client operation fails.
OnRead	Occurs when data is available to be read from the client.
OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
OnWrite	Occurs when data can be written to the client.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the TelnetClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient Constructor

Initializes a new instance of the TelnetClient class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public TelnetClient();
```

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient Properties

The properties of the **TelnetClient** class are listed below. For a complete list of **TelnetClient** class members, see the TelnetClient Members topic.

Public Instance Properties

AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
Blocking	Gets and sets a value which indicates if the client is in blocking mode.
CertificateExpires	Get a value that specifies the date that the security certificate expires.
CertificateIssued	Get a value that specifies the date that the security certificate was issued.
CertificateIssuer	Get a value that provides information about the organization that issued the certificate.
CertificateName	Gets and sets a value that specifies the name of the client certificate.
CertificatePassword	Gets and sets the password associated with the client certificate.
CertificateStatus	Gets a value which indicates the status of the security certificate returned by the remote host.
CertificateStore	Gets and sets a value that specifies the name of the local certificate store.
CertificateSubject	Gets a value that provides information about the organization that the server certificate was issued to.
CertificateUser	Gets and sets the user that owns the client certificate.
CipherStrength	Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.
CodePage	Gets and sets the code page used when reading and writing text.
Handle	Gets a value that specifies the client handle allocated for the current session.
HashStrength	Gets a value which specifies the length of the message digest that was selected for a secure connection.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.

IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsConnected	Gets a value which indicates if a connection to the remote host has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsReadable	Gets a value which indicates if there is data available to be read from the socket connection to the server.
IsThere	Gets a value which indicates if the the remote host is responsive
IsWritable	Gets a value which indicates if data can be written to the client without blocking.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets the local Internet address that the client is bound to.
LocalEcho	Enables and disables the echoing of characters by the remote host.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
Options	Gets and sets a value which specifies one or more client options.
Password	Gets and sets the password used to authenticate the client session.
RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.
Secure	Gets and sets a value which specifies if a secure connection is established.
SecureCipher	Gets a value that specifies the encryption algorithm used for a secure connection.
SecureHash	Gets a value that specifies the message digest algorithm used for a secure connection.
SecureKeyExchange	Gets a value that specifies the key exchange algorithm used for a secure connection.
SecureProtocol	Gets and sets a value which specifies the protocol used for a secure connection.

Status	Gets a value which specifies the current status of the client.
Terminal	Gets and sets the terminal type used for a remote login session.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
UserName	Gets and sets the username used to authenticate the client session.
Version	Gets a value which returns the current version of the TelnetClient class library.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.AutoResolve Property

Gets and sets a value that determines if host names and addresses are automatically resolved.

[Visual Basic]

```
Public Property AutoResolve As Boolean
```

[C#]

```
public bool AutoResolve {get; set;}
```

Property Value

Returns **true** if host names are automatically resolved to Internet addresses. The default value is **false**.

Remarks

Setting the **AutoResolve** property determines if the class automatically resolves host names and addresses specified by the **HostName** and **HostAddress** properties. If set to **true**, setting the **HostName** property will cause the class to automatically determine the corresponding IP address and update the **HostAddress** property accordingly. Likewise, setting the **HostAddress** property will cause the class to determine the host name and update the **HostName** property. Setting this property to **false** prevents the class from resolving host names until a connection attempt is made.

It is important to note that setting the **HostName** or **HostAddress** property may cause the current thread to block, sometimes for several seconds, until the name or address is resolved. To prevent this behavior, set this property value to **false**.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Blocking Property

Gets and sets a value which indicates if the client is in blocking mode.

[Visual Basic]

```
Public Property Blocking As Boolean
```

[C#]

```
public bool Blocking {get; set;}
```

Property Value

Returns **true** if the client is in blocking mode; otherwise it returns **false**. The default value is **true**.

Remarks

Setting the **Blocking** property determines if client operations complete synchronously or asynchronously. If set to **true**, then each client operation (such as sending or receiving data) will return when the operation has completed or timed-out. If set to **false**, client operations will return immediately. If the operation would result in the client blocking (such as attempting to read data when no data has been sent by the remote host), an error is generated.

It is important to note that certain events, such as **OnDisconnect, OnRead** and **OnWrite** are only fired if the client is in non-blocking mode.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.CertificateExpires Property

Get a value that specifies the date that the security certificate expires.

[Visual Basic]
```
Public ReadOnly Property CertificateExpires As String
```

[C#]
```
public string CertificateExpires {get;}
```

Property Value

A string which specifies a date using the local date and time format.

Remarks

The **CertificateExpires** property returns a string that specifies the date and time that the security certificate expires. This property will return an empty string if a secure connection has not been established with the remote host.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.CertificateIssued Property

Get a value that specifies the date that the security certificate was issued.

[Visual Basic]

```
Public ReadOnly Property CertificateIssued As String
```

[C#]

```
public string CertificateIssued {get;}
```

Property Value

A string which specifies a date using the local date and time format.

Remarks

The **CertificateIssued** property returns a string that specifies the date and time that the security certificate was issued. This property will return an empty string if a secure connection has not been established with the remote host.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.CertificateIssuer Property

Get a value that provides information about the organization that issued the certificate.

[Visual Basic]

```
Public ReadOnly Property CertificateIssuer As String
```

[C#]

```
public string CertificateIssuer {get;}
```

Property Value

A string that contains a comma separated list of name value pairs.

Remarks

The **CertificateIssuer** property returns a string that contains information about the organization that issued the server certificate. The string value is a comma separated list of tagged name and value pairs. In the nomenclature of the X.500 standard, each of these pairs are called a relative distinguished name (RDN), and when concatenated together, forms the issuer's distinguished name (DN). For example:

```
C=US, O="RSA Data Security, Inc.", OU=Secure Server Certification Authority
```

To obtain a specific value, such as the name of the issuer or the issuer's country, the application must parse the string returned by this property. Some of the common tokens used in the distinguished name are:

Token	Description
C	The ISO standard two character country code.
S	The name of the state or province.
L	The name of the city or locality.
O	The name of the company or organization.
OU	The name of the department or organizational unit
CN	The common name; with X.509 certificates, this is the domain name of the site the certificate was issued for.

This property will return an empty string if a secure connection has not been established with the remote host.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.CertificateName Property

Gets and sets a value that specifies the name of the client certificate.

[Visual Basic]

```
Public Property CertificateName As String
```

[C#]

```
public string CertificateName {get; set;}
```

Property Value

A string which specifies the certificate name.

Remarks

The **CertificateName** property is used to specify the name of a client certificate to use when establishing a secure connection. It is only required that you set this property value if the server requires a client certificate for authentication. If this property is not set, a client certificate will not be provided to the server. If a certificate name is specified, the certificate must have a private key associated with it, otherwise the connection attempt will fail because the control will be unable to create a security context for the session.

When the certificate store is searched for a matching certificate, it will first search for any certificate with a friendly name that matches the property value. If no valid certificate is found, it will then search for a certificate with a matching common name.

Certificates may be installed and viewed on the local system using the Certificate Manager that is included with the Windows operating system. For more information, refer to the documentation for the Microsoft Management Console.

See Also

TelnetClient Class | SocketTools Namespace | CertificateStore Property | Secure Property

TelnetClient.CertificateStatus Property

Gets a value which indicates the status of the security certificate returned by the remote host.

[Visual Basic]
```
Public ReadOnly Property CertificateStatus As SecurityCertificate
```

[C#]
```
public TelnetClient.SecurityCertificate CertificateStatus {get;}
```

Property Value

A SecurityCertificate enumeration value which specifies the status of the certificate.

Remarks

The **CertificateStatus** property is used to determine the status of the security certificate returned by the remote host when a secure connection has been established. This property value should be checked after the connection to the server has completed, but prior to beginning a transaction.

Note that if the certificate cannot be validated, the secure connection will not be automatically terminated. It is the responsibility of your application to determine the best course of action to take if the certificate is invalid. Even if the security certificate cannot be validated, the data exchanged with the remote host will still be encrypted.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.CertificateStore Property

Gets and sets a value that specifies the name of the local certificate store.

[Visual Basic]
```
Public Property CertificateStore As String
```

[C#]
```
public string CertificateStore {get; set;}
```

Property Value

A string which specifies the certificate store name. The default value is the current user's personal certificate store.

Remarks

The **CertificateStore** property is used to specify the name of the certificate store which contains the security certificate to use when establishing a secure connection. The certificate may either be stored in the registry or in a file. If the certificate is stored in the registry, then this property should be set to one of the following predefined values:

Store Name	Description
CA	Certification authority certificates. These are certificates that are issued by entities which are entrusted to issue certificates to other individuals or organizations. Companies such as VeriSign and Thawte act as certification authorities.
MY	Personal certificates and their associated private keys for the current user. This store typically holds the client certificates used to establish a user's credentials. If a certificate store is not specified, this is the default value that is used.
ROOT	Certificates that have been self-signed by a certificate authority. Root certificates for a number of different certification authorities such as VeriSign and Thawte are installed as part of the operating system and periodically updated by Microsoft.

In most cases the client certificate will be installed in the user's personal certificate store, and therefore it is not necessary to set this property value because that is the default location that will be used to search for the certificate. This property is only used if the **CertificateName** property is also set to a valid certificate name.

If you are using a local certificate store, with the certificate and private key stored in the registry, you can explicitly specify whether the certificate store for the current user or the local machine (all users) should be used. This is done by prefixing the certificate store name with "HKCU" for the current user, or "HKLM" for the local machine. For example, a certificate store name of "HKLM:MY" would specify the personal certificate store for the local machine, rather than the current user. If neither prefix is specified, it will default to the certificate store for the current user.

This property may also be used to specify a file that contains the client certificate. In this case, the property should specify the full path to the file and must contain both the certificate and private key in PKCS12

format. If the file is protected by a password, the **CertificatePassword** property must also be set to specify the password.

See Also

TelnetClient Class | SocketTools Namespace | CertificatePassword Property | Secure Property

TelnetClient.CertificateSubject Property

Gets a value that provides information about the organization that the server certificate was issued to.

[Visual Basic]

```
Public ReadOnly Property CertificateSubject As String
```

[C#]

```
public string CertificateSubject {get;}
```

Property Value

A string that contains a comma separated list of name value pairs.

Remarks

The **CertificateSubject** property returns a string that contains information about the organization that the server certificate was issued to. The string value is a comma separated list of tagged name and value pairs. In the nomenclature of the X.500 standard, each of these pairs are called a relative distinguished name (RDN), and when concatenated together, forms the issuer's distinguished name (DN). For example:

```
C=US, O="RSA Data Security, Inc.", OU=Secure Server Certification Authority
```

To obtain a specific value, such as the name of the issuer or the issuer's country, the application must parse the string returned by this property. Some of the common tokens used in the distinguished name are:

Token	Description
C	The ISO standard two character country code.
S	The name of the state or province.
L	The name of the city or locality.
O	The name of the company or organization.
OU	The name of the department or organizational unit
CN	The common name; with X.509 certificates, this is the domain name of the site the certificate was issued for.

This property will return an empty string if a secure connection has not been established with the remote host.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.CipherStrength Property

Gets a value that indicates the length of the key used by the encryption algorithm for a secure connection.

[Visual Basic]

```
Public ReadOnly Property CipherStrength As Integer
```

[C#]

```
public int CipherStrength {get;}
```

Property Value

An integer value which specifies the encryption key length if a secure connection has been established; otherwise a value of 0 is returned.

Remarks

The **CipherStrength** property returns the number of bits in the key used to encrypt the secure data stream. Common values returned by this property are 128 and 256. A key length of 40 or 56 bits is considered insecure and subject to brute force attacks. 128-bit and 256-bit keys are considered secure. If this property returns a value of 0, this means that a secure connection has not been established with the remote host.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.CodePage Property

Gets and sets the code page used when reading and writing text.

[Visual Basic]

```
Public Property CodePage As Integer
```

[C#]

```
public int CodePage {get; set;}
```

Property Value

An integer value which specifies the current code page. A value of zero specifies the default code page for the current locale should be used. To preserve the original Unicode text, you can use code page 65001 which specifies UTF-8 character encoding.

Remarks

All data which is exchanged over a socket is sent and received as 8-bit bytes, typically referred to as "octets" in networking terminology. However, strings in .NET are Unicode where each character is represented by 16 bits. To send and receive data using strings, these Unicode strings are converted to a stream of bytes.

By default, strings are converted to an array of bytes using the code page for the current locale, mapping the 16-bit Unicode characters to bytes. Similarly, when reading data from the socket into a string buffer, the stream of bytes received from the remote host are converted to Unicode before they are returned to your application.

If you are exchanging text with another system and it appears to corrupted or characters are being replaced with question marks or other symbols, it is likely the system is sending text which is using a different character encoding. Most services use UTF-8 encoding to represent non-ASCII characters and selecting the UTF-8 code page will typically resolve the issue.

Strings are only guaranteed to be safe when sending and receiving text. Using a string data type is not recommended when reading or writing binary data to a socket. If possible, you should always use a byte array as the buffer parameter for the Read and Write methods whenever you are exchanging binary data.

For backwards compatibility, this class defaults to using the code page for the current locale. This property value directly corresponds to Windows code page identifiers, and will accept any valid code page supported by the .NET Framework. Setting this property to an invalid code page will generate an exception.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Handle Property

Gets a value that specifies the client handle allocated for the current session.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer which represents a client handle. If there is no active connection, a value of -1 is returned.

Remarks

The **Handle** property specifies the numeric descriptor of the current client session.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.HashStrength Property

Gets a value which specifies the length of the message digest that was selected for a secure connection.

[Visual Basic]

```
Public ReadOnly Property HashStrength As Integer
```

[C#]

```
public int HashStrength {get;}
```

Property Value

An integer value which specifies the length of the message digest if a secure connection has been established; otherwise a value of 0 is returned.

Remarks

The **HashStrength** property returns the number of bits used in the message digest (hash) that was selected. Common values returned by this property are 128 and 160. If this property returns a value of 0, this means that a secure connection has not been established with the remote host.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.HostAddress Property

Gets and sets a value which specifies the Internet address used to establish a connection.

[Visual Basic]

```
Public Property HostAddress As String
```

[C#]

```
public string HostAddress {get; set;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **HostAddress** property can be used to set the Internet address for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the address is assigned to a valid host name, the **HostName** property will be updated with that value.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.HostName Property

Gets and sets a value which specifies the host name used to establish a connection.

[Visual Basic]

```
Public Property HostName As String
```

[C#]

```
public string HostName {get; set;}
```

Property Value

A string which specifies a host name.

Remarks

The **HostName** property can be used to set the host name for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the name can be resolved to a valid Internet address, the **HostAddress** property will be updated with that value.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.IsBlocked Property

Gets a value which indicates if the current thread is performing a blocking client operation.

[Visual Basic]

```
Public ReadOnly Property IsBlocked As Boolean
```

[C#]

```
public bool IsBlocked {get;}
```

Property Value

Returns **true** if the current thread is blocking, otherwise returns **false**.

Remarks

The **IsBlocked** property returns **true** if the current thread is blocked performing an operation. Because the Windows Sockets API only permits one blocking operation per thread of execution, this property should be checked before starting any blocking operation in response to an event.

If the **IsBlocked** property returns **false**, this means there are no blocking operations on the current thread at that time. However, this does not guarantee that the next client operation will not fail. An application should always check the return value from a client operation and check the value of the **LastError** property if an error occurs.

Note that this property will return **true** if there is any blocking operation being performed by the current thread, regardless of whether this specific instance of the class is responsible for the blocking operation or not.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.IsConnected Property

Gets a value which indicates if a connection to the remote host has been established.

[Visual Basic]

```
Public ReadOnly Property IsConnected As Boolean
```

[C#]

```
public bool IsConnected {get;}
```

Property Value

Returns **true** if the connection has been established; otherwise returns **false**.

Remarks

The **IsConnected** property can only be used to indicate if there is still a logical connection to the remote host. It cannot be used to detect abnormal conditions such as the remote host aborting the connection, the physical network connection being lost or other critical errors.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.IsInitialized Property

Gets a value which indicates if the current instance of the class has been initialized successfully.

[Visual Basic]
```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]
```
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the class instance has been initialized; otherwise returns **false**.

Remarks

The **IsInitialized** property is used to determine if the current instance of the class has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the class may not be able to initialize itself.

The most common reasons that a class instance may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required networking libraries or not being able to access the system registry.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.IsReadable Property

Gets a value which indicates if there is data available to be read from the socket connection to the server.

[Visual Basic]

```
Public ReadOnly Property IsReadable As Boolean
```

[C#]

```
public bool IsReadable {get;}
```

Property Value

Returns **true** if there is data available to be read; otherwise returns **false**.

Remarks

The **IsReadable** property returns **true** if data can be read from the client without blocking. For non-blocking sessions, this property can be checked before the application attempts to read the client. Note that even if this property does return **true** indicating that there is data available to be read, applications should always check the return value from the **Read** method.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.IsThere Property

Gets a value which indicates if the the remote host is responsive

[Visual Basic]

```
Public ReadOnly Property IsThere As Boolean
```

[C#]

```
public bool IsThere {get;}
```

Property Value

Returns **true** if the connection is active and the remote host is responsive to the client; otherwise returns **false**.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.IsWritable Property

Gets a value which indicates if data can be written to the client without blocking.

[Visual Basic]

```
Public ReadOnly Property IsWritable As Boolean
```

[C#]

```
public bool IsWritable {get;}
```

Property Value

Returns **true** if data can be written to the client; otherwise returns **false**.

Remarks

The **IsWritable** property returns **true** if data can be written to the client without blocking. For non-blocking sessions, this property can be checked before the application attempts to write data to the client. Note that even if this property does return **true** indicating that data can be written to the client, applications should always check the return value from the **Write** method.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public TelnetClient.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.LocalAddress Property

Gets the local Internet address that the client is bound to.

[Visual Basic]

```
Public ReadOnly Property LocalAddress As String
```

[C#]

```
public string LocalAddress {get;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **LocalAddress** property returns the local Internet address that the client is bound to when a connection is established with a remote host. This property may return either an IPv4 or IPv6 formatted address, depending on the type of connection that was established.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.LocalEcho Property

Enables and disables the echoing of characters by the remote host.

[Visual Basic]

```
Public Property LocalEcho As Boolean
```

[C#]

```
public bool LocalEcho {get; set;}
```

Property Value

A boolean value which specifies if local echo has been enabled.

Remarks

The **LocalEcho** property enables or disables the echoing of characters by the remote host. If set to **true**, the remote host will be instructed to not echo characters, making the client responsible for displaying user input. The default value for this property is **false**, specifying that the server should echo user input back to the client.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.LocalName Property

Gets a value which specifies the host name for the local system.

[Visual Basic]

```
Public ReadOnly Property LocalName As String
```

[C#]

```
public string LocalName {get;}
```

Property Value

A string which specifies the local host name.

Remarks

The **LocalName** property returns the fully-qualified host name assigned to the local system. If the system has not been configured with an Internet domain name, then this property will return the NetBIOS name assigned to the local system.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.LocalPort Property

Gets the local port number the client is bound to.

[Visual Basic]

```
Public ReadOnly Property LocalPort As Integer
```

[C#]

```
public int LocalPort {get;}
```

Property Value

An integer value which specifies a port number. The default value is 0.

Remarks

The **LocalPort** property is used to identify the local port number that the client is bound to to when a connection is established with a remote host.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Options Property

Gets and sets a value which specifies one or more client options.

[Visual Basic]

```
Public Property Options As TelnetOptions
```

[C#]

```
public TelnetClient.TelnetOptions Options {get; set;}
```

Property Value

Returns one or more TelnetOptions enumeration flags which specify the options for the client. The default value for this property is **clientOptionNone**.

Remarks

The **Options** property specifies one or more default options options which are used when establishing a connection using the **Connect** method.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Password Property

Gets and sets the password used to authenticate the client session.

[Visual Basic]

```
Public Property Password As String
```

[C#]

```
public string Password {get; set;}
```

Property Value

A string which specifies the password.

Remarks

If a password is not specified when the **Login** method is called, the value of this property will be used as the default password when authenticating the client session.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.RemotePort Property

Gets and sets a value which specifies the remote port number.

[Visual Basic]

```
Public Property RemotePort As Integer
```

[C#]

```
public int RemotePort {get; set;}
```

Property Value

An integer value which specifies a port number.

Remarks

The **RemotePort** property is used to set the port number that will be used to establish a connection with a remote host. If the port number specifies a well-known port, the **RemoteService** property will be updated with that name.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Secure Property

Gets and sets a value which specifies if a secure connection is established.

[Visual Basic]

```
Public Property Secure As Boolean
```

[C#]

```
public bool Secure {get; set;}
```

Property Value

Returns **true** if a secure connection is established; otherwise returns **false**. The default value is **false**.

Remarks

The **Secure** property determines if a secure connection is established with the remote host. The default value for this property is **false**, which specifies that a standard connection to the server is used. To establish a secure connection, the application should set this property value to **true** prior to calling the **Connect** method. Once the connection has been established, the client may exchange data with the server as with standard connections.

It is strongly recommended that any application that sets this property **true** use error handling to trap an errors that may occur. If the control is unable to initialize the security libraries, or otherwise cannot create a secure session for the client, an exception may be generated when this property value is set.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.SecureCipher Property

Gets a value that specifies the encryption algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureCipher As SecureCipherAlgorithm
```

[C#]

```
public TelnetClient.SecureCipherAlgorithm SecureCipher {get;}
```

Property Value

A SecureCipherAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureCipher** property returns a value which identifies the algorithm used to encrypt the data stream. If a secure connection has not been established, this property will return a value of **cipherNone**.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.SecureHash Property

Gets a value that specifies the message digest algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureHash As SecureHashAlgorithm
```

[C#]

```
public TelnetClient.SecureHashAlgorithm SecureHash {get;}
```

Property Value

A SecureHashAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureHash** property returns a value which identifies the message digest (hash) algorithm that was selected when a secure connection was established. If a secure connection has not been established, this property will return a value of **hashNone**.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.SecureKeyExchange Property

Gets a value that specifies the key exchange algorithm used for a secure connection.

[Visual Basic]

```
Public ReadOnly Property SecureKeyExchange As SecureKeyAlgorithm
```

[C#]

```
public TelnetClient.SecureKeyAlgorithm SecureKeyExchange {get;}
```

Property Value

A SecureKeyAlgorithm enumeration value which identifies the algorithm.

Remarks

The **SecureKeyExchange** property returns a value which identifies the key exchange algorithm that was selected when a secure connection was established. If a secure connection has not been established, this property will return a value of **keyExchangeNone**.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.SecureProtocol Property

Gets and sets a value which specifies the protocol used for a secure connection.

[Visual Basic]
```
Public Property SecureProtocol As SecurityProtocols
```

[C#]
```
public TelnetClient.SecurityProtocols SecureProtocol {get; set;}
```

Property Value

A SecurityProtocols enumeration value which identifies the protocol to be used when establishing a secure connection.

Remarks

The **SecureProtocol** property can be used to specify the security protocol to be used when establishing a secure connection with a server or accepting a secure connection from a client. By default, the class will attempt to use either SSL v3 or TLS v1 to establish the connection, with the appropriate protocol automatically selected based on the capabilities of the remote host. It is recommended that you only change this property value if you fully understand the implications of doing so. Assigning a value to this property will override the default protocol and force the class to attempt to use only the protocol specified.

Multiple security protocols may be specified by combining them using a bitwise **or** operator. After a connection has been established, this property will identify the protocol that was selected. Attempting to set this property after a connection has been established will result in an exception being thrown. This property should only be set after setting the **Secure** property to **true** and before calling the **Accept** or **Connect** methods.

In some cases, a server may only accept a secure connection if the TLS v1 protocol is specified. If the security protocol is not compatible with the server, then the connection will fail with an error indicating that the control is unable to establish a security context for the session. In this case, try assigning the property to **protocolTLS1** and attempt the connection again.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Status Property

Gets a value which specifies the current status of the client.

[Visual Basic]

```
Public ReadOnly Property Status As TelnetStatus
```

[C#]

```
public TelnetClient.TelnetStatus Status {get;}
```

Property Value

A TelnetStatus enumeration value which specifies the current client status.

Remarks

The **Status** property returns the current status of the client. This property can be used to check on blocking connections to determine if the client is interacting with the remote host before taking some action.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Terminal Property

Gets and sets the terminal type used for a remote login session.

[Visual Basic]

```
Public Property Terminal As String
```

[C#]

```
public string Terminal {get; set;}
```

Property Value

A string which specifies the terminal type.

Remarks

The **Terminal** property specifies the terminal type of the remote host for display purposes. On UNIX based systems, the terminal name corresponds to a termcap or terminfo entry as set in the TERM environment variable. On Windows based systems which implement the rlogin service, this property may be ignored and the server will assume that the client is capable of displaying ANSI escape sequences. On VMS systems, the terminal name should correspond to the terminal type used with the SET TERMINAL/DEVICE command.

If this property is set to an empty string and no terminal type is specified when the **Login** method is called, a default terminal type named "unknown" will be used. On most UNIX and VMS systems this defines a terminal which is not capable of cursor positioning using control or escape sequences. This terminal type may not be recognized and an error may be displayed when the user logs in indicating that the terminal type is invalid.

Refer to the documentation for the server system to determine what terminal type names are available to you. Remember that on UNIX systems, the terminal type is case-sensitive. Some of the more common terminal types are:

Terminal	Description
ansi	This terminal type is usually available on UNIX based servers. This specifies that the client is capable of displaying standard ANSI escape sequences for cursor control.
dumb	This terminal type typically specifies a terminal display which does not support control or escape sequences for cursor positioning. If you do not want escape sequences embedded in the data stream and the server returns an error if the terminal type is not specified, try using this terminal type.
pcansi	This terminal type is usually available on UNIX based servers. This specifies that the client is a using a PC terminal emulator that supports basic ANSI escape sequences for cursor control. This may also enable escape sequences which can set the display colors.
vt100	This terminal type is usually available on UNIX and VMS based servers. On some VMS systems this

	string may need to be specified as DEC-VT100. This specifies that the client is capable of emulating a DEC VT100 terminal. The VT100 supports many of the same cursor control sequences as an ANSI terminal.
vt220	This terminal type is usually available on UNIX and VMS based servers. On some VMS systems this string may need to be specified as DEC-VT220. This specifies that the client is capable of emulating a DEC VT220 terminal, which is a later version of the VT100.
vt320	This terminal type is usually available on UNIX and VMS based servers. On some VMS systems this string may need to be specified as DEC-VT320. This specifies that the client is capable of emulating a DEC VT320 terminal, which is similar to the VT100 and VT220 and provides advanced features such as the ability to set display colors.
xterm	This terminal type is may be available on UNIX based servers which have X Windows installed. This specifies that the client is a using the X Windows xterm emulator which supports standard ANSI escape sequences for cursor control.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.ThreadModel Property

Gets and sets a value which specifies the threading model for the class instance.

[Visual Basic]

```
Public Property ThreadModel As ThreadingModel
```

[C#]

```
public TelnetClient.ThreadingModel ThreadModel {get; set;}
```

Property Value

Returns one or more ThreadingModel enumeration value which specifies the threading model for the client. The default value for this property is **modelSingleThread**.

Remarks

The **ThreadModel** property specifies the threading model that is used by the class instance when a connection is established. The default value for this property is **modelSingleThread**, which specifies that only the thread that established the connection should be permitted to invoke methods. It is important to note that this threading model does not limit the application to a single thread of execution. When a session is established using the **Connect** method, that session is attached to the thread that created it. From that point on, until the session is terminated, only the owner may invoke methods in that instance of the class. The ownership of the class instance may be transferred from one thread to another using the **AttachThread** method.

Setting this property to **modelFreeThread** disables certain internal safety checks that are performed by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads. The application must ensure that no two threads will attempt to invoke a blocking method at the same time. In other words, if one thread invokes a method, the application must ensure that another thread will not attempt to invoke any other method at the same time using the same instance of the class.

Changing the value of this property will not affect an active client session. The threading model must be specified prior to invoking the **Connect** method.

See Also

TelnetClient Class | SocketTools Namespace | AttachThread Method | ThreadingModel Enumeration | ThreadModel Attribute

TelnetClient.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]
```
Public Property ThrowError As Boolean
```

[C#]
```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Timeout Property

Gets and sets a value which specifies a timeout period in seconds.

[Visual Basic]

```
Public Property Timeout As Integer
```

[C#]

```
public int Timeout {get; set;}
```

Property Value

An integer value which specifies a timeout period in seconds.

Remarks

Setting the **Timeout** property specifies the number of seconds until a blocking operation fails and returns an error.

The timeout period is only used when the client is in blocking mode. Although this property can be changed when the client is in non-blocking mode, the value will be ignored until the client is returned to blocking mode.

For most applications it is recommended the timeout period be set between 10 and 20 seconds.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Trace Property

Gets and sets a value which indicates if network function logging is enabled.

[Visual Basic]
```
Public Property Trace As Boolean
```

[C#]
```
public bool Trace {get; set;}
```

Property Value

Returns **true** if network function tracing is enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Trace** property is used to enable (or disable) the tracing of network function calls. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the SocketTools networking classes and is enabled or disabled for an entire process. This means that once trace logging is enabled for a given component, all of the function calls made by the process using any of the SocketTools classes will be logged. For example, if you have an application using both the File Transfer Protocol and Post Office Protocol classes, and you set the **Trace** property to **true**, function calls made by both classes will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all classes used by the process.

If trace logging is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the logfile is fairly large. Since logfiles can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

When redistributing your application, make sure that you include the **SocketTools10.TraceLog.dll** module with your installation. If this library is not present, then no trace output will be generated and the value of the **Trace** property will be ignored. Only those function calls made by the SocketTools networking classes will be logged. Calls made directly to the Windows Sockets API, or calls made by other classes, will not be logged.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.TraceFile Property

Gets and sets a value which specifies the name of the logfile.

[Visual Basic]

```
Public Property TraceFile As String
```

[C#]

```
public string TraceFile {get; set;}
```

Property Value

A string which specifies the name of the file.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named **SocketTools.log** is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFile** property should be set to the same value for each instance. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
 MyApp INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0 MyApp WRN:
connect(46, 192.0.0.1:1234, 16) returned -1 [10035] MyApp ERR: accept(46,
NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the control is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Note that if the specified file cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.TraceFlags Property

Gets and sets a value which specifies the client function tracing flags.

[Visual Basic]

```
Public Property TraceFlags As TraceOptions
```

[C#]

```
public TelnetClient.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration which specifies the amount of detail written to the trace logfile.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Because network function tracing is enabled per-process, the trace flags are shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFlags** property should be set to the same value for each instance. Changing the trace flags for any one instance of the class will affect the logging performed for all SocketTools classes used by the application.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being written and the error **errorOperationWouldBlock** occurs, a warning is generated because the application simply needs to attempt to write the data at a later time.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.UserName Property

Gets and sets the username used to authenticate the client session.

[Visual Basic]

```
Public Property UserName As String
```

[C#]

```
public string UserName {get; set;}
```

Property Value

A string which specifies the username.

Remarks

If a username is not specified when the **Login** method is called, the value of this property will be used as the default username when authenticating the client session.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Version Property

Gets a value which returns the current version of the TelnetClient class library.

[Visual Basic]
```
Public ReadOnly Property Version As String
```

[C#]
```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the TelnetClient class library. This value can be used by an application for validation and debugging purposes.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient Methods

The methods of the **TelnetClient** class are listed below. For a complete list of **TelnetClient** class members, see the TelnetClient Members topic.

Public Instance Methods

Abort	Aborts the current session and terminates the connection.
AttachThread	Attach an instance of the class to the current thread
Break	Sends a break signal to the remote host.
Cancel	Cancel the current blocking client operation.
Connect	Overloaded. Establish a connection with a remote host.
Disconnect	Terminate the connection with a remote host.
Dispose	Overloaded. Releases all resources used by TelnetClient.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the TelnetClient class.
Login	Overloaded. Login to the remote server.
Read	Overloaded. Read data from the server and store it in a byte array.
ReadLine	Overloaded. Read up to a line of data from the server and return it in a string buffer.
Reset	Reset the internal state of the object, resetting all properties to their default values.
Search	Overloaded. Search for a specific character sequence in the data stream.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
Write	Overloaded. Write one or more bytes of data to the server.
WriteLine	Overloaded. Send a line of text to the server, terminated by a carriage-return and linefeed.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the TelnetClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Abort Method

Aborts the current session and terminates the connection.

[Visual Basic]

```
Public Function Abort() As Boolean
```

[C#]

```
public bool Abort();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Abort** method sends an abort sequence to the server and the connection to the server is terminated. Once this method returns, the client is no longer connected to the server. If a program is currently executing on the server at the time this method is called, that program may be terminated as a result of the session being aborted. Applications should normally call the **Disconnect** method to gracefully disconnect from the server and should only use this method when the connection must be aborted immediately.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.AttachThread Method

Attach an instance of the class to the current thread

[Visual Basic]

```
Public Function AttachThread() As Boolean
```

[C#]

```
public bool AttachThread();
```

Return Value

A boolean value which specifies if the client could be attached to the current thread. If this method returns **false**, the client could not be attached to the thread and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

When an instance of the class is created it is associated with the current thread that created it. Normally, if another thread attempts to perform an operation using that instance, an error is returned. This is used to ensure that other threads cannot interfere with an operation being performed by the owner thread. In some cases, it may be desirable for one thread in an application to create an instance of the class, establish a connection and then pass that instance to another worker thread. The **AttachThread** method can be used to change the ownership of the class instance to the new worker thread.

This method should be called by the new thread immediately after it has been created, and if the new thread does not release the handle itself, the ownership of the handle should be restored by the original thread. Under no circumstances should **AttachThread** be used to forcibly destroy an instance of a class allocated by another thread while a blocking operation is in progress. To cancel a blocking operation, use the **Cancel** method and then delete the class instance after the blocking function exits and control is returned to the current thread.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Break Method

Sends a break signal to the remote host.

[Visual Basic]
```
Public Function Break() As Boolean
```

[C#]
```
public bool Break();
```

Remarks

The **Break** method sends a signal to the server which may terminate an application that is currently running. The actual response to the break signal depends on the application.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Cancel Method

Cancel the current blocking client operation.

[Visual Basic]

```
Public Sub Cancel()
```

[C#]

```
public void Cancel();
```

Remarks

When the **Cancel** method is called, the blocking client operation will not immediately fail. An internal flag is set which causes the blocking operation to exit with an error. This means that the application cannot cancel an operation and immediately perform some other blocking function. Instead it must allow the calling stack to unwind, returning back to the blocking operation before making any further function calls.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Connect Method

Establish a connection with a remote host.

Overload List

Establish a connection with a remote host.

public bool Connect();

Establish a connection with a remote host.

public bool Connect(string);

Establish a connection with a remote host.

public bool Connect(string,int);

Establish a connection with a remote host.

public bool Connect(string,int,int);

Establish a connection with a remote host.

public bool Connect(string,int,int,TelnetOptions);

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Connect Method ()

Establish a connection with a remote host.

[Visual Basic]
```
Overloads Public Function Connect() As Boolean
```

[C#]
```
public bool Connect();
```

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection using assigned property values.

The value of the **HostName** or **HostAddress** property will be used to determine the host name or address to connect to.

The value of the **RemotePort** property will be used to determine the port number to connect to.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Connect Overload List

TelnetClient.Connect Method (String)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Connect Overload List

TelnetClient.Connect Method (String, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort
);
```

Parameters

hostName
A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort
An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Connect Overload List

TelnetClient.Connect Method (String, Int32, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   int timeout
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Connect Overload List

TelnetClient.Connect Method (String, Int32, Int32, TelnetOptions)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer, _
   ByVal options As TelnetOptions _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   int timeout,
   TelnetOptions options
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

options

One or more of the TelnetOptions enumeration flags.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Connect Overload List

TelnetClient.Disconnect Method

Terminate the connection with a remote host.

[Visual Basic]

```
Public Sub Disconnect()
```

[C#]

```
public void Disconnect();
```

Remarks

The **Disconnect** method terminates the connection with the remote host and releases the client handle allocated by the class. Note that the socket is not immediately released when the connection is terminated and will enter a wait state for two minutes. After the time wait period has elapsed, the client will be released by the operating system. This is a normal safety mechanism to handle any packets that may arrive after the connection has been closed.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Dispose Method

Releases all resources used by TelnetClient.

Overload List

Releases all resources used by TelnetClient.

public void Dispose();

Releases the unmanaged resources allocated by the TelnetClient class and optionally releases the managed resources.

protected virtual void Dispose(bool);

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Dispose Method ()

Releases all resources used by TelnetClient.

[Visual Basic]

```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]

```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Dispose Overload List

TelnetClient.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the TelnetClient class and optionally releases the managed resources.

[Visual Basic]

```
Overridable Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]

```
protected virtual void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **TelnetClient** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Dispose Overload List

TelnetClient.Finalize Method

Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.

[Visual Basic]

```
Overrides Protected Sub Finalize()
```

[C#]

```
protected override void Finalize();
```

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Initialize Method

Initialize an instance of the TelnetClient class.

Overload List

Initialize an instance of the TelnetClient class.

public bool Initialize();

Initialize an instance of the TelnetClient class.

public bool Initialize(string);

See Also

TelnetClient Class | SocketTools Namespace | Uninitialize Method

TelnetClient.Initialize Method ()

Initialize an instance of the TelnetClient class.

[Visual Basic]

```
Overloads Public Function Initialize() As Boolean
```

[C#]

```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the TelnetClient class, loading the networking library and allocating resources for the current thread. Typically it is not necessary to explicitly call this method because the instance of the class is initialized by the class constructor. However, if the **Uninitialize** method is called, the class must be re-initialized before any other methods are called.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Initialize Overload List | Uninitialize Method

TelnetClient.Initialize Method (String)

Initialize an instance of the TelnetClient class.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the TelnetClient class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the TelnetClient class.

Example

The following example shows how to use the Initialize method to initialize an instance of the class. This example assumes that the license key string has been defined in code.

```
SocketTools.TelnetClient telnetClient = new SocketTools.TelnetClient();

if (telnetClient.Initialize(strLicenseKey) == false)
{
    MessageBox.Show(telnetClient.LastErrorString, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    return;
}
```

```
Dim telnetClient As New SocketTools.TelnetClient

If telnetClient.Initialize(strLicenseKey) = False Then
    MsgBox(telnetClient.LastErrorString, vbIconExclamation)
    Exit Sub
End If
```

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

TelnetClient.Login Method

Login to the remote server.

Overload List

Login to the remote server.

public bool Login();

Login to the remote server.

public bool Login(string,string);

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Login Method ()

Login to the remote server.

[Visual Basic]
```
Overloads Public Function Login() As Boolean
```

[C#]
```
public bool Login();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Login** method identifies the user to the remote server. The value of the **UserName** and **Password** properties will be used to authenticate the client session. If the user name or password is invalid, an error will occur. By default, when a connection is established, the user is automatically authenticated. This method is typically used if you wish to log in as another user during the same session.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Login Overload List

TelnetClient.Login Method (String, String)

Login to the remote server.

[Visual Basic]

```
Overloads Public Function Login( _
   ByVal userName As String, _
   ByVal userPassword As String _
) As Boolean
```

[C#]

```
public bool Login(
   string userName,
   string userPassword
);
```

Parameters

userName

A string that specifies the name of the user logging into the server.

userPassword

A string that specifies the password used to authenticate the user.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Login** method identifies the user to the remote server. If the user name or password is invalid, an error will occur. By default, when a connection is established, the **UserName**, **Password** and **Account** properties are used to automatically log the user in to the server. This method is typically used if you wish to log in as another user during the same session.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Login Overload List

TelnetClient.Read Method

Read data from the server and store it in a byte array.

Overload List

Read data from the server and store it in a byte array.

public int Read(byte[]);

Read data from the server and store it in a byte array.

public int Read(byte[],int);

Read data from the server and store it in a string.

public int Read(ref string);

Read data from the server and store it in a string.

public int Read(ref string,int);

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Read Method (Byte[])

Read data from the server and store it in a byte array.

[Visual Basic]

```
Overloads Public Function Read( _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Read(
   byte[] buffer
);
```

Parameters

buffer

A byte array that the data will be stored in.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the size of the byte array passed to the method. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Read Overload List

TelnetClient.Read Method (Byte[], Int32)

Read data from the server and store it in a byte array.

[Visual Basic]

```
Overloads Public Function Read( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Read(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array that the data will be stored in.

length

An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the number of bytes specified. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Read Overload List

TelnetClient.Read Method (String)

Read data from the server and store it in a string.

[Visual Basic]

```
Overloads Public Function Read( _
   ByRef buffer As String _
) As Integer
```

[C#]

```
public int Read(
   ref string buffer
);
```

Parameters

buffer

A string that will contain the data read from the client.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to a maximum of 4096 bytes. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Read Overload List

TelnetClient.Read Method (String, Int32)

Read data from the server and store it in a string.

[Visual Basic]

```
Overloads Public Function Read( _
   ByRef buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Read(
   ref string buffer,
   int length
);
```

Parameters

buffer
A string that will contain the data read from the client.

length
An integer value which specifies the maximum number of bytes of data to read.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the number of bytes specified. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Read Overload List

TelnetClient.ReadLine Method

Read up to a line of data from the server and return it in a string buffer.

Overload List

Read up to a line of data from the server and return it in a string buffer.

public bool ReadLine(ref string);

Read up to a line of data from the server and return it in a string buffer.

public bool ReadLine(ref string,int);

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.ReadLine Method (String)

Read up to a line of data from the server and return it in a string buffer.

[Visual Basic]

```
Overloads Public Function ReadLine( _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool ReadLine(
   ref string buffer
);
```

Parameters

buffer

A string which will contain the data read from the socket.

Return Value

This method returns a Boolean value which specifies if a line of data has been read. A value of **true** indicates a line of data has been read. If an error occurs or there is no more data available to read, then the method will return **false**. It is possible for data to be returned in the string buffer even if the return value is **false**. Applications should check the length of the string after the method returns to determine if any data was copied into the buffer. For example, if a timeout occurs while the method is waiting for more data to arrive on the socket, it will return zero; however, data may have already been copied into the string buffer prior to the error condition. It is the responsibility of the application to process that data, regardless of the method return value.

Remarks

The **ReadLine** method reads data from the server up to 4096 bytes in length or until an end-of-line character sequence is encountered. Unlike the **Read** method which reads arbitrary bytes of data, this method is specifically designed to return a single line of text data in a string variable. When an end-of-line character sequence is encountered, the method will stop and return the data up to that point; the string will not contain the carriage-return or linefeed characters.

There are some limitations when using the **ReadLine** method. The method should only be used to read text, never binary data. In particular, it will discard nulls, linefeed and carriage return control characters. This method will force the current thread to block until an end-of-line character sequence is processed, the read operation times out or the remote host closes its end of the socket connection. If the **Blocking** property is set to **false**, calling this method will automatically switch the client into a blocking mode, read the data and then restore the client to non-blocking mode. If another network operation is attempted while **ReadLine** is blocked waiting for data from the remote host, an error will occur. It is recommended that this method only be used with blocking connections.

The **Read** and **ReadLine** methods can be intermixed, however be aware that the **Read** method will consume any data that has already been buffered by the **ReadLine** method and this may have unexpected results.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.ReadLine Overload List

TelnetClient.ReadLine Method (String, Int32)

Read up to a line of data from the server and return it in a string buffer.

[Visual Basic]

```
Overloads Public Function ReadLine( _
   ByRef buffer As String, _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool ReadLine(
   ref string buffer,
   int length
);
```

Parameters

buffer

A string which will contain the data read from the socket.

length

An integer value which specifies the maximum number of bytes of data to read.

Return Value

This method returns a Boolean value which specifies if a line of data has been read. A value of **true** indicates a line of data has been read. If an error occurs or there is no more data available to read, then the method will return **false**. It is possible for data to be returned in the string buffer even if the return value is **false**. Applications should check the length of the string after the method returns to determine if any data was copied into the buffer. For example, if a timeout occurs while the method is waiting for more data to arrive on the socket, it will return zero; however, data may have already been copied into the string buffer prior to the error condition. It is the responsibility of the application to process that data, regardless of the method return value.

Remarks

The **ReadLine** method reads data from the server up to the specified number of bytes or until an end-of-line character sequence is encountered. Unlike the **Read** method which reads arbitrary bytes of data, this method is specifically designed to return a single line of text data in a string variable. When an end-of-line character sequence is encountered, the method will stop and return the data up to that point; the string will not contain the carriage-return or linefeed characters.

There are some limitations when using the **ReadLine** method. The method should only be used to read text, never binary data. In particular, it will discard nulls, linefeed and carriage return control characters. This method will force the current thread to block until an end-of-line character sequence is processed, the read operation times out or the remote host closes its end of the socket connection. If the **Blocking** property is set to **false**, calling this method will automatically switch the client into a blocking mode, read the data and then restore the client to non-blocking mode. If another network operation is attempted while **ReadLine** is blocked waiting for data from the remote host, an error will occur. It is recommended that this method only be used with blocking connections.

The **Read** and **ReadLine** methods can be intermixed, however be aware that the **Read** method will consume any data that has already been buffered by the **ReadLine** method and this may have unexpected results.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.ReadLine Overload List

TelnetClient.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. If a socket has been allocated, it will be released and any active connections will be terminated. All properties will be reset to their default values.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Search Method

Search for a specific character sequence in the data stream.

Overload List

Search for a specific character sequence in the data stream.

public bool Search(string);

Search for a specific character sequence in the data stream.

public bool Search(string,byte[],ref int);

Search for a specific character sequence in the data stream.

public bool Search(string,ref string);

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Search Method (String)

Search for a specific character sequence in the data stream.

[Visual Basic]

```
Overloads Public Function Search( _
   ByVal value As String _
) As Boolean
```

[C#]

```
public bool Search(
   string value
);
```

Parameters

value

A string argument which specifies the sequence of characters to search for in the data stream. When this sequence of characters is found, the method will return.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Search** method searches for a character sequence in the data stream and stops reading when it is found. This is useful when the client wants to automate responses to the server, such as executing a command and processing the output. The method will discard any data received up to and including the specified character sequence.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Search Overload List

TelnetClient.Search Method (String, Byte[], Int32)

Search for a specific character sequence in the data stream.

[Visual Basic]

```
Overloads Public Function Search( _
   ByVal value As String, _
   ByVal buffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool Search(
   string value,
   byte[] buffer,
   ref int length
);
```

Parameters

value
A string argument which specifies the sequence of characters to search for in the data stream. When this sequence of characters is found, the method will return.

buffer
An byte array that will contain the output sent by the server, up to and including the search string character sequence.

length
An integer value passed by reference which should be initialized to the maximum number of bytes of data to store in the buffer. When the method returns, this value will be updated with the actual number of bytes stored in the buffer.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Search** method searches for a character sequence in the data stream and stops reading when it is found. This is useful when the client wants to automate responses to the server, such as executing a command and processing the output. The method collects the output from the server and stores it in a buffer provided by the caller. When the method returns, the buffer will contain everything sent by the server up to and including the search string.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Search Overload List

TelnetClient.Search Method (String, String)

Search for a specific character sequence in the data stream.

[Visual Basic]

```
Overloads Public Function Search( _
   ByVal value As String, _
   ByRef buffer As String _
) As Boolean
```

[C#]

```
public bool Search(
   string value,
   ref string buffer
);
```

Parameters

value

A string argument which specifies the sequence of characters to search for in the data stream. When this sequence of characters is found, the method will return.

buffer

An string that will contain the output sent by the server, up to and including the search string character sequence.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Search** method searches for a character sequence in the data stream and stops reading when it is found. This is useful when the client wants to automate responses to the server, such as executing a command and processing the output. The method collects the output from the server and stores it in a buffer provided by the caller. When the method returns, the buffer will contain everything sent by the server up to and including the search string.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Search Overload List

TelnetClient.Uninitialize Method

Uninitialize the class library and release any resources allocated for the current thread.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method terminates any active connection, releases resources allocated for the current thread and unloads the networking library. After this method has been called, no further client operations may be performed until the class instance has been re-initialized.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the class is no longer needed.

See Also

TelnetClient Class | SocketTools Namespace | Initialize Method

TelnetClient.Write Method

Write one or more bytes of data to the server.

Overload List

Write one or more bytes of data to the server.

public int Write(byte[]);

Write one or more bytes of data to the server.

public int Write(byte[],int);

Write a character to the server.

public int Write(char);

Write one or more characters to the server.

public int Write(char,int);

Write a string of characters to the server.

public int Write(string);

Write a string of characters to the server.

public int Write(string,int);

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.Write Method (Byte[])

Write one or more bytes of data to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Write(
   byte[] buffer
);
```

Parameters

buffer

A byte array that contains the data to be written to the server.

Return Value

An integer value which specifies the number of bytes actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Write Overload List

TelnetClient.Write Method (Byte[], Int32)

Write one or more bytes of data to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Write(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array that contains the data to be written to the server.

length

An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more bytes of data to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Write Overload List

TelnetClient.Write Method (Char)

Write a character to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal key As Char _
) As Integer
```

[C#]

```
public int Write(
   char key
);
```

Parameters

key

A character which will be written to the server.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one character to the server. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Write Overload List

TelnetClient.Write Method (Char, Int32)

Write one or more characters to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal key As Char, _
   ByVal repeat As Integer _
) As Integer
```

[C#]

```
public int Write(
   char key,
   int repeat
);
```

Parameters

key

A character which will be written to the server.

repeat

The number of characters that will be written to the server.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends one or more characters to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Write Overload List

TelnetClient.Write Method (String)

Write a string of characters to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String _
) As Integer
```

[C#]

```
public int Write(
   string buffer
);
```

Parameters

buffer

A string which contains the data to be written to the server.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

The string will be converted to an array of bytes before being written to the socket. By default, the character encoding used will be for the current locale. Depending on the contents of the string, the number of bytes written may be different than the string length specified. This is because the conversion from Unicode to a byte array may result in a multi-byte character sequence.

You should never use strings to read and write binary data. Always use byte arrays to ensure that no character conversion is performed.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Write Overload List

TelnetClient.Write Method (String, Int32)

Write a string of characters to the server.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String, _
   ByVal length As Integer _
) As Integer
```

[C#]

```
public int Write(
   string buffer,
   int length
);
```

Parameters

buffer
A string which contains the data to be written to the server.

length
An integer value which specifies the maximum number of characters to write. This value cannot be larger than the length of the string specified by the caller.

Return Value

An integer value which specifies the number of characters actually written to the server. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Write** method sends a string of characters to the server. If there is enough room in the client's internal send buffer to accommodate all of the data, it is copied to the send buffer and control immediately returns to the caller. If amount of data exceeds the available buffer space and the client is in blocking mode, then the method will block until the data can be sent. If the client is in non-blocking mode and the send buffer is full, an error will occur.

The string will be converted to an array of bytes before being written to the socket. By default, the character encoding used will be for the current locale. Depending on the contents of the string, the number of bytes written may be different than the string length specified. This is because the conversion from Unicode to a byte array may result in a multi-byte character sequence.

You should never use strings to read and write binary data. Always use byte arrays to ensure that no character conversion is performed.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.Write Overload List

TelnetClient.WriteLine Method

Send an empty line of text to the server, terminated by a carriage-return and linefeed.

Overload List

Send an empty line of text to the server, terminated by a carriage-return and linefeed.

public bool WriteLine();

Send a line of text to the server, terminated by a carriage-return and linefeed.

public bool WriteLine(string);

Send a line of text to the server, terminated by a carriage-return and linefeed.

public bool WriteLine(string,ref int);

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.WriteLine Method ()

Send an empty line of text to the server, terminated by a carriage-return and linefeed.

[Visual Basic]

```
Overloads Public Function WriteLine() As Boolean
```

[C#]

```
public bool WriteLine();
```

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

The **WriteLine** method will send an empty line of text, terminated by a carriage-return and linefeed. Calling this method will force the application to block until the complete line of text has been written, the write operation times out or the remote host aborts the connection. If this method is called with the **Blocking** property set to **false**, it will automatically switch the client into a blocking mode, send the data and then restore the client to non-blocking mode. If another network operation is attempted while the **WriteLine** method is blocked sending data to the remote host, an error will occur. It is recommended that this method only be used with blocking connections.

The **Write** and **WriteLine** methods can be safely intermixed.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.WriteLine Overload List

TelnetClient.WriteLine Method (String)

Send a line of text to the server, terminated by a carriage-return and linefeed.

[Visual Basic]

```
Overloads Public Function WriteLine( _
   ByVal buffer As String _
) As Boolean
```

[C#]

```
public bool WriteLine(
   string buffer
);
```

Parameters

buffer

A string which contains the data that will be sent to the remote host. The data will always be terminated with a carriage-return and linefeed control character sequence. If the string is empty, then a only a carriage-return and linefeed are written to the socket. Note that if the string contains a null character, any data that follows the null character will be discarded.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

The **WriteLine** method should only be used to send text, never binary data. In particular, this method will discard any data that follows a null character and will append linefeed and carriage return control characters to the data stream. Calling this method will force the current thread to block until the complete line of text has been written, the write operation times out or the remote host aborts the connection. If this method is called with the **Blocking** property set to **false**, it will automatically switch the client into a blocking mode, send the data and then restore the client to non-blocking mode. If another network operation is attempted while the **WriteLine** method is blocked sending data to the remote host, an error will occur. It is recommended that this method only be used with blocking connections.

The string will be converted to an array of bytes before being written to the socket. By default, the character encoding used will be for the current locale. Depending on the contents of the string, the number of bytes written may be different than the string length specified. This is because the conversion from Unicode to a byte array may result in a multi-byte character sequence.

You should never use strings to read and write binary data. Always use byte arrays to ensure that no character conversion is performed.

The **Write** and **WriteLine** methods can be safely intermixed.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.WriteLine Overload List

TelnetClient.WriteLine Method (String, Int32)

Send a line of text to the server, terminated by a carriage-return and linefeed.

[Visual Basic]

```
Overloads Public Function WriteLine( _
   ByVal buffer As String, _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool WriteLine(
   string buffer,
   ref int length
);
```

Parameters

buffer

A string which contains the data that will be sent to the remote host. The data will always be terminated with a carriage-return and linefeed control character sequence. If the string is empty, then a only a carriage-return and linefeed are written to the socket. Note that if the string contains a null character, any data that follows the null character will be discarded.

length

An integer value which specifies the maximum number of characters to write. This value cannot be larger than the length of the string specified by the caller.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred. To get extended error information, check the value of the **LastError** property.

Remarks

The **WriteLine** method should only be used to send text, never binary data. In particular, this method will discard any data that follows a null character and will append linefeed and carriage return control characters to the data stream. Calling this method will force the current thread to block until the complete line of text has been written, the write operation times out or the remote host aborts the connection. If this method is called with the **Blocking** property set to **false**, it will automatically switch the client into a blocking mode, send the data and then restore the client to non-blocking mode. If another network operation is attempted while the **WriteLine** method is blocked sending data to the remote host, an error will occur. It is recommended that this method only be used with blocking connections.

The string will be converted to an array of bytes before being written to the socket. By default, the character encoding used will be for the current locale. Depending on the contents of the string, the number of bytes written may be different than the string length specified. This is because the conversion from Unicode to a byte array may result in a multi-byte character sequence.

You should never use strings to read and write binary data. Always use byte arrays to ensure that no character conversion is performed.

The **Write** and **WriteLine** methods can be safely intermixed.

See Also

TelnetClient Class | SocketTools Namespace | TelnetClient.WriteLine Overload List

TelnetClient Events

The events of the **TelnetClient** class are listed below. For a complete list of **TelnetClient** class members, see the TelnetClient Members topic.

Public Instance Events

⚡ OnCancel	Occurs when a blocking client operation is canceled.
⚡ OnConnect	Occurs when a connection is established with the remote host.
⚡ OnDisconnect	Occurs when the remote host disconnects from the local system.
⚡ OnError	Occurs when an client operation fails.
⚡ OnRead	Occurs when data is available to be read from the client.
⚡ OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
⚡ OnWrite	Occurs when data can be written to the client.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.OnCancel Event

Occurs when a blocking client operation is canceled.

[Visual Basic]

```
Public Event OnCancel As EventHandler
```

[C#]

```
public event EventHandler OnCancel;
```

Remarks

The **OnCancel** event is generated when a blocking client operation, such as sending or receiving data, is canceled with the **Cancel** method. To assist in determining which operation was canceled, check the value of the **Status** property.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.OnConnect Event

Occurs when a connection is established with the remote host.

[Visual Basic]

```
Public Event OnConnect As EventHandler
```

[C#]

```
public event EventHandler OnConnect;
```

Remarks

The **OnConnect** event occurs when a connection is made with a remote host as a result of a **Connect** method call. When the **Connect** method is called and the **Blocking** property is set to **false**, a socket is created but the connection is not actually established until after this event occurs. Between the time connection process is started and this event fires, no operation may be performed on the client other than calling the **Disconnect** method.

This event is only generated if the client is in non-blocking mode.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.OnDisconnect Event

Occurs when the remote host disconnects from the local system.

[Visual Basic]

```
Public Event OnDisconnect As EventHandler
```

[C#]

```
public event EventHandler OnDisconnect;
```

Remarks

The **OnDisconnect** event occurs when the remote host closes its connection, terminating the client session with the application. Because there may still be data in the client receive buffers, you should continue to read data from the client until the **Read** method returns a value of 0. Once all of the data has been read, you should call the **Disconnect** method to close the local socket and release the resources allocated for the client.

This event is only generated if the client is in non-blocking mode.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.OnError Event

Occurs when an client operation fails.

[Visual Basic]

```
Public Event OnError As OnErrorEventHandler
```

[C#]

```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type TelnetClient.ErrorEventArgs containing data related to this event. The following **TelnetClient.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a client operation fails.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see TelnetClient.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.TelnetClient.ErrorEventArgs

[Visual Basic]

```
Public Class TelnetClient.ErrorEventArgs
    Inherits EventArgs
```

[C#]

```
public class TelnetClient.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.TelnetClient (in SocketTools.TelnetClient.dll)

See Also

TelnetClient.ErrorEventArgs Members | SocketTools Namespace

TelnetClient.ErrorEventArgs Members

TelnetClient.ErrorEventArgs overview

Public Instance Constructors

TelnetClient.ErrorEventArgs Constructor	Initializes a new instance of the TelnetClient.ErrorEventArgs class.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

TelnetClient.ErrorEventArgs Class | SocketTools Namespace

TelnetClient.ErrorEventArgs Constructor

Initializes a new instance of the TelnetClient.ErrorEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public TelnetClient.ErrorEventArgs();
```

See Also

TelnetClient.ErrorEventArgs Class | SocketTools Namespace

TelnetClient.ErrorEventArgs Properties

The properties of the **TelnetClient.ErrorEventArgs** class are listed below. For a complete list of **TelnetClient.ErrorEventArgs** class members, see the TelnetClient.ErrorEventArgs Members topic.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

TelnetClient.ErrorEventArgs Class | SocketTools Namespace

TelnetClient.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

TelnetClient.ErrorEventArgs Class | SocketTools Namespace | Error Property

TelnetClient.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public TelnetClient.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

TelnetClient.ErrorEventArgs Class | SocketTools Namespace | Description Property

TelnetClient.OnRead Event

Occurs when data is available to be read from the client.

[Visual Basic]

```
Public Event OnRead As EventHandler
```

[C#]

```
public event EventHandler OnRead;
```

Remarks

The **OnRead** event occurs when data is available to be read from the client. This event is level-triggered, which means that once this event fires, it will not occur again until some data has been read from the client. This design prevents an application from being flooded with event notifications. It is recommended that your application read all of the available data from the server and store it in a local buffer for processing. See the example below.

This event is only generated if the client is in non-blocking mode.

Example

```
Private Sub Socket_OnRead(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Socket.OnRead
    Dim strBuffer As String
    Dim nRead As Integer

    Do
        ' Read up to m_nBufferSize bytes of data from the server
        nRead = Socket.Read(strBuffer, m_nBufferSize)

        If nRead > 0 Then
            ' Append the data to an internal buffer for processing
            m_dataBuffer = m_dataBuffer + strBuffer
        End If
    Loop Until nRead < 1

    ProcessData()
End Sub
```

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.OnTimeout Event

Occurs when a blocking operation fails to complete before the timeout period elapses.

[Visual Basic]

```
Public Event OnTimeout As EventHandler
```

[C#]

```
public event EventHandler OnTimeout;
```

Remarks

The **OnTimeout** event occurs when a blocking operation, such as sending or receiving data on the client, fails to complete before the specified timeout period elapses. The timeout period for a blocking operation can be adjusted by setting the **Timeout** property.

This event is only generated if the client is in blocking mode.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.OnWrite Event

Occurs when data can be written to the client.

[Visual Basic]

```
Public Event OnWrite As EventHandler
```

[C#]

```
public event EventHandler OnWrite;
```

Remarks

The **OnWrite** event occurs when the application can write data to the client. This event will typically occur when a connection is first established with the remote host, and after the **Write** method has failed because there was insufficient memory available in the client send buffers. In the second case, when some of the buffered data has been successfully sent to the remote host and there is space available in the send buffers, this event is used to signal the application that it may attempt to send more data.

This event is only generated if the client is in non-blocking mode.

See Also

TelnetClient Class | SocketTools Namespace

TelnetClient.ErrorCode Enumeration

Specifies the error codes returned by the TelnetClient class.

[Visual Basic]
```
Public Enum TelnetClient.ErrorCode
```

[C#]
```
public enum TelnetClient.ErrorCode
```

Remarks

The TelnetClient class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.
errorProxyAuthenticationRequired	Proxy authentication required.

errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.
errorUnsupportedMediaType	Request specified an unsupported media type.

errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.
errorNoMoreMailboxes	No more mailboxes exist on this server.

errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.
errorInvalidThreadHandle	The specified thread handle is no longer valid.

errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been defined.

errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or unrecognized.

errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.

Requirements

Namespace: SocketTools

Assembly: SocketTools.TelnetClient (in SocketTools.TelnetClient.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

TelnetClient.SecureCipherAlgorithm Enumeration

Specifies the encryption algorithms that the TelnetClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum TelnetClient.SecureCipherAlgorithm
```

[C#]

```
[Flags]
public enum TelnetClient.SecureCipherAlgorithm
```

Remarks

The TelnetClient class uses the **SecureCipherAlgorithm** enumeration to identify which encryption algorithm was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
cipherNone	No cipher has been selected. A secure connection has not been established with the remote host.	0
cipherRC2	The RC2 block cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.	1
cipherRC4	The RC4 stream cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.	2
cipherRC5	The RC5 block cipher was selected. This is a variable key length cipher which supports keys up to 2040 bits, in 8-bit increments.	4
cipherDES	The DES (Data Encryption Standard) block cipher was selected. This is a fixed key length cipher using 56-bit keys.	8
cipherDES3	The Triple DES block cipher was selected. This cipher encrypts the data three times using different keys, effectively using a 168-bit key length.	16
cipherDESX	A variant of the DES block cipher which XORs an extra 64-bits of the key before and after the plaintext has been encrypted, increasing the key size to 184 bits.	32
cipherAES	The Advanced Encryption Standard	64

	cipher (also known as the Rijndael cipher) is a fixed block size cipher which use a key size of 128, 192 or 256 bits. This cipher is supported on Windows XP SP3 SP3 and later versions of the operating system.	
cipherSkipjack	The Skipjack block cipher was selected. This is a fixed key length cipher, using 80-bit keys.	128

Requirements

Namespace: SocketTools

Assembly: SocketTools.TelnetClient (in SocketTools.TelnetClient.dll)

See Also

SocketTools Namespace

TelnetClient.SecureHashAlgorithm Enumeration

Specifies the hash algorithms that the TelnetClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum TelnetClient.SecureHashAlgorithm
```

[C#]

```
[Flags]
public enum TelnetClient.SecureHashAlgorithm
```

Remarks

The TelnetClient class uses the **SecureHashAlgorithm** enumeration to identify the message digest (hash) algorithm that was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
hashNone	No hash algorithm has been selected. This is not a secure connection with the server.	0
hashMD5	The MD5 algorithm was selected. This algorithm produces a 128-bit message digest. This algorithm is no longer considered to be cryptographically secure.	1
hashSHA	The SHA-1 algorithm was selected. This algorithm produces a 160-bit message digest. This algorithm is no longer considered to be cryptographically secure.	2
hashSHA256	The SHA-256 algorithm was selected. This algorithm produces a 256-bit message digest.	4
hashSHA384	The SHA-384 algorithm was selected. This algorithm produces a 384-bit message digest.	8
hashSHA512	The SHA-512 algorithm was selected. This algorithm produces a 512-bit message digest.	16

Requirements

Namespace: SocketTools

Assembly: SocketTools.TelnetClient (in SocketTools.TelnetClient.dll)

See Also

SocketTools Namespace

TelnetClient.SecureKeyAlgorithm Enumeration

Specifies the key exchange algorithms that the TelnetClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum TelnetClient.SecureKeyAlgorithm
```

[C#]
```
[Flags]
public enum TelnetClient.SecureKeyAlgorithm
```

Remarks

The TelnetClient class uses the **SecureKeyAlgorithm** enumeration to identify the key exchange algorithm that was selected when a secure connection was established with the remote host.

Members

Member Name	Description	Value
keyExchangeNone	No key exchange algorithm has been selected. This is not a secure connection with the server.	0
keyExchangeRSA	The RSA public key exchange algorithm has been selected.	1
keyExchangeKEA	The KEA public key exchange algorithm has been selected. This is an improved version of the Diffie-Hellman public key algorithm.	2
keyExchangeDH	The Diffie-Hellman public key exchange algorithm has been selected.	4
keyExchangeECDH	The Elliptic Curve Diffie-Hellman key exchange algorithm was selected. This is a variant of the Diffie-Hellman algorithm which uses elliptic curve cryptography. This key exchange algorithm is only supported on Windows XP SP3 SP3 and later versions of the operating system.	8

Requirements

Namespace: SocketTools

Assembly: SocketTools.TelnetClient (in SocketTools.TelnetClient.dll)

See Also

SocketTools Namespace

TelnetClient.SecurityCertificate Enumeration

Specifies the security certificate status values that may be returned by the TelnetClient class.

[Visual Basic]
```
Public Enum TelnetClient.SecurityCertificate
```

[C#]
```
public enum TelnetClient.SecurityCertificate
```

Remarks

The TelnetClient class uses the **SecurityCertificate** enumeration to identify the current status of the certificate that was provided by the remote host when a secure connection was established.

Members

Member Name	Description
certificateNone	No certificate information is available. A secure connection was not established with the server.
certificateValid	The certificate is valid.
certificateNoMatch	The certificate is valid, however the domain name specified in the certificate does not match the domain name of the remote host. The application can examine the **CertificateSubject** property to determine the site the certificate was issued to.
certificateExpired	The certificate has expired and is no longer valid. The application can examine the **CertificateExpires** property to determine when the certificate expired.
certificateRevoked	The certificate has been revoked and is no longer valid. It is recommended that the application immediately terminate the connection if this status is returned.
certificateUntrusted	The certificate has not been issued by a trusted authority, or the certificate is not trusted on the local host. It is recommended that the application immediately terminate the connection if this status is returned.
certificateInvalid	The certificate is invalid. This typically indicates that the internal structure of the certificate is damaged. It is recommended that the application immediately terminate the connection if this status is returned.

Requirements

Namespace: SocketTools

Assembly: SocketTools.TelnetClient (in SocketTools.TelnetClient.dll)

See Also

SocketTools Namespace

TelnetClient.SecurityProtocols Enumeration

Specifies the security protocols that the TelnetClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum TelnetClient.SecurityProtocols
```

[C#]

```
[Flags]
public enum TelnetClient.SecurityProtocols
```

Remarks

The TelnetClient class uses the **SecurityProtocols** enumeration to specify one or more security protocols to be used when establishing a connection with a remote host. Multiple protocols may be specified if necessary and the actual protocol used will be negotiated with the remote host. It is recommended that most applications use **protocolDefault** when creating a secure connection.

Members

Member Name	Description	Value
protocolNone	No security protocol will be used, a secure connection will not be established.	0
protocolSSL2	The SSL 2.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections.	1
protocolSSL3	The SSL 3.0 protocol has been selected. This protocol has been deprecated and is no longer widely used. It is not recommended that this protocol be used when establishing secure connections. In most cases, this protocol is only selected if TLS is not supported by the server.	2
protocolTLS10	The TLS 1.0 protocol has been selected. This version of the protocol is commonly used by older servers and is the only version of TLS supported on Windows platforms prior to Windows 7 SP1 and Windows Server 2008 R2.	4
protocolTLS11	The TLS 1.1 protocol has been selected. This version of TLS is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of the	8

	operating system.	
protocolTLS12	The TLS 1.2 protocol has been selected. This is the default version of the protocol and is supported on Windows 7 SP1 and Windows Server 2008 R2 and later versions of Windows. It is recommended that you use this version of TLS.	16
protocolSSL	Any version of the Secure Sockets Layer (SSL) protocol should be used. The actual protocol version used will be negotiated with the remote host.	3
protocolTLS	Any version of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with with preference for TLS 1.2.	28
protocolTLS1	Version 1.0, 1.1 or 1.2 of the the Transport Layer Security (TLS) protocol should be used. The actual protocol version used will be negotiated with the remote host, with preference for TLS 1.2.	28
protocolDefault	The default selection of security protocols will be used when establishing a connection. The TLS 1.2, 1.1 and 1.0 protocols will be negotiated with the host, in that order of preference. This option will always request the latest version of the preferred security protocols and is the recommended value.	16
protocolUnknown	An unknown or unsupported security protocol has been specified. This value indicates an error condition.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.TelnetClient (in SocketTools.TelnetClient.dll)

See Also

SocketTools Namespace

TelnetClient.TelnetOptions Enumeration

Specifies the options that the TelnetClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum TelnetClient.TelnetOptions
```

[C#]
```
[Flags]
public enum TelnetClient.TelnetOptions
```

Remarks

The TelnetClient class uses the **TelnetOptions** enumeration to specify one or more options to be used when establishing a connection with a remote host. Multiple options may be specified if necessary.

Members

Member Name	Description	Value
optionNone	No option specified.	0
optionDefault	The default connection option. This is the same as specifying **optionNone**.	0
optionTunnel	This option specifies that a tunneled TCP connection and/or port-forwarding is being used to establish the connection to the server. This changes the behavior of the client with regards to internal checks of the destination IP address and remote port number, default capability selection and how the connection is established. This option also forces all connections to be outbound and enables the firewall compatibility features in the client.	1024
optionTrustedSite	This option specifies the server is trusted. The server certificate will not be validated and the connection will always be permitted. This option only affects connections using either the SSL or TLS protocols.	2048
optionSecure	The default option for establishing a secure connection. This option is the same as specifying **optionSecureImplicit**.	4096
optionSecureImplicit	This option specifies the client should attempt to establish a secure session immediately after connecting to the server. Note that the server must	4096

	support secure connections using either the SSL or TLS protocol.	
optionSecureExplicit	This option specifies the client should attempt to establish a secure connection with the server using standard Telnet option negotiation. This requires that the server support the START_TLS option.	8192
optionSecureFallback	This option specifies the client should permit the use of less secure cipher suites for compatibility with legacy servers. If this option is specified, the client will permit connections using TLS 1.0 and cipher suites that use RC4, MD5 and SHA1.	32768
optionPreferIPv6	This option specifies the client should prefer the use of IPv6 if the server hostname can be resolved to both an IPv6 and IPv4 address. This option is ignored if the local system does not have IPv6 enabled, or when the hostname can only be resolved to an IPv4 address. If the server hostname can only be resolved to an IPv6 address, the client will attempt to establish a connection using IPv6 regardless if this option has been specified.	262144
optionFreeThread	This option specifies that class methods may be called from any thread, and not only the thread that established the connection. Using this option disables certain internal safety checks that are made by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads.	524288

Requirements

Namespace: SocketTools

Assembly: SocketTools.TelnetClient (in SocketTools.TelnetClient.dll)

See Also

SocketTools Namespace

TelnetClient.TelnetStatus Enumeration

Specifies the status values that may be returned by the TelnetClient class.

[Visual Basic]
```
Public Enum TelnetClient.TelnetStatus
```

[C#]
```
public enum TelnetClient.TelnetStatus
```

Remarks

The TelnetClient class uses the **TelnetStatus** enumeration to identify the current status of the client.

Members

Member Name	Description
statusUnused	A client session has not been created. Attempts to perform any network operations, such as sending or receiving data, will generate an error.
statusIdle	A client session has been created, but is not currently in use. A blocking socket operation can be executed at this point.
statusConnect	The client is in the process of establishing a connection with a remote host.
statusRead	The client is in the process of receiving data from a remote host.
statusWrite	The client is in the process of sending data to a remote host.
statusDisconnect	The client session is being closed and subsequent attempts to access the client will result in an error.

Requirements

Namespace: SocketTools

Assembly: SocketTools.TelnetClient (in SocketTools.TelnetClient.dll)

See Also

SocketTools Namespace

TelnetClient.TraceOptions Enumeration

Specifies the logging options that the TelnetClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum TelnetClient.TraceOptions
```

[C#]

```
[Flags]
public enum TelnetClient.TraceOptions
```

Remarks

The TelnetClient class uses the **TraceOptions** enumeration to specify what kind of debugging information is written to the trace logfile. These options are only meaningful when trace logging is enabled by setting the **Trace** property to **true**.

Members

Member Name	Description	Value
traceDefault	The default trace logging option. This is the same as specifying the **traceInfo** option.	0
traceInfo	All network function calls are written to the trace file. This is the default value.	0
traceError	Only those network function calls which fail are recorded in the trace file.	1
traceWarning	Only those network function calls which fail, or return values which indicate a warning, are recorded in the trace file.	2
traceHexDump	All network function calls are written to the trace file, plus all the data that is sent or received is displayed, in both ASCII and hexadecimal format.	4
traceProcess	All function calls in the current process are logged, rather than only those functions in the current thread. This option is useful for multithreaded applications that are using worker threads.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.TelnetClient (in SocketTools.TelnetClient.dll)

See Also

SocketTools Namespace

TelnetClient.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub TelnetClient.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void TelnetClient.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ErrorEventArgs that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.TelnetClient (in SocketTools.TelnetClient.dll)

See Also

SocketTools Namespace

TelnetClient.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see TelnetClient.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.TelnetClient.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class TelnetClient.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class TelnetClient.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the TelnetClient class.

Example

```
<Assembly: SocketTools.TelnetClient.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.TelnetClient.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.TelnetClient (in SocketTools.TelnetClient.dll)

See Also

TelnetClient.RuntimeLicenseAttribute Members | SocketTools Namespace

TelnetClient.RuntimeLicenseAttribute Members

TelnetClient.RuntimeLicenseAttribute overview

Public Instance Constructors

TelnetClient.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

TelnetClient.RuntimeLicenseAttribute Class | SocketTools Namespace

TelnetClient.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public TelnetClient.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the TelnetClient class.

See Also

TelnetClient.RuntimeLicenseAttribute Class | SocketTools Namespace

TelnetClient.RuntimeLicenseAttribute Properties

The properties of the **TelnetClient.RuntimeLicenseAttribute** class are listed below. For a complete list of **TelnetClient.RuntimeLicenseAttribute** class members, see the TelnetClient.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

TelnetClient.RuntimeLicenseAttribute Class | SocketTools Namespace

TelnetClient.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

TelnetClient.RuntimeLicenseAttribute Class | SocketTools Namespace

TelnetClientException Class

The exception that is thrown when a client error occurs.

For a list of all members of this type, see TelnetClientException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.TelnetClientException

[Visual Basic]

```
Public Class TelnetClientException
    Inherits ApplicationException
```

[C#]

```
public class TelnetClientException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A TelnetClientException is thrown by the TelnetClient class when an error occurs.

The default constructor for the TelnetClientException class sets the **ErrorCode** property to the last client error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.TelnetClient (in SocketTools.TelnetClient.dll)

See Also

TelnetClientException Members | SocketTools Namespace

TelnetClientException Members

TelnetClientException overview

Public Instance Constructors

TelnetClientException	Overloaded. Initializes a new instance of the TelnetClientException class.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

TelnetClientException Class | SocketTools Namespace

TelnetClientException Constructor

Initializes a new instance of the TelnetClientException class with the last network error code.

Overload List

Initializes a new instance of the TelnetClientException class with the last network error code.

public TelnetClientException();

Initializes a new instance of the TelnetClientException class with a specified error number.

public TelnetClientException(int);

Initializes a new instance of the TelnetClientException class with a specified error message.

public TelnetClientException(string);

Initializes a new instance of the TelnetClientException class with a specified error message and a reference to the inner exception that is the cause of this exception.

public TelnetClientException(string,Exception);

See Also

TelnetClientException Class | SocketTools Namespace

TelnetClientException Constructor ()

Initializes a new instance of the TelnetClientException class with the last network error code.

[Visual Basic]

```
Overloads Public Sub New()
```

[C#]

```
public TelnetClientException();
```

Remarks

The ctor constructor sets the **ErrorCode** property to the last client error that occurred. For more information about the errors that may occur, refer to the TelnetClient.ErrorCode enumeration.

See Also

TelnetClientException Class | SocketTools Namespace | TelnetClientException Constructor Overload List

TelnetClientException Constructor (String)

Initializes a new instance of the TelnetClientException class with a specified error message.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]

```
public TelnetClientException(
   string message
);
```

Parameters

message

The error message that explains the reason for the exception.

Remarks

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

TelnetClientException Class | SocketTools Namespace | TelnetClientException Constructor Overload List

TelnetClientException Constructor (String, Exception)

Initializes a new instance of the TelnetClientException class with a specified error message and a reference to the inner exception that is the cause of this exception.

[Visual Basic]
```
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal innerException As Exception _
)
```

[C#]
```
public TelnetClientException(
   string message,
   Exception innerException
);
```

Parameters

message
The error message that explains the reason for the exception.

innerException
The exception that is the cause of the current exception. If the ***innerException*** parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

TelnetClientException Class | SocketTools Namespace | TelnetClientException Constructor Overload List

TelnetClientException Constructor (Int32)

Initializes a new instance of the TelnetClientException class with a specified error number.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

[C#]

```
public TelnetClientException(
   int code
);
```

Parameters

code

An integer value which specifies an error code.

Remarks

This constructor sets the **ErrorCode** property to the specified error code. For more information about the errors that may occur, refer to the TelnetClient.ErrorCode enumeration.

See Also

TelnetClientException Class | SocketTools Namespace | TelnetClientException Constructor Overload List

TelnetClientException Properties

The properties of the **TelnetClientException** class are listed below. For a complete list of **TelnetClientException** class members, see the TelnetClientException Members topic.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

TelnetClientException Class | SocketTools Namespace

TelnetClientException.ErrorCode Property

Gets a value which specifies the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property ErrorCode As ErrorCode
```

[C#]

```
public TelnetClient.ErrorCode ErrorCode {get;}
```

Property Value

Returns a TelnetClient.ErrorCode enumeration value which specifies the error code.

Remarks

The **ErrorCode** property returns the error code that specifies the cause of the exception.

The default constructor for the TelnetClientException class sets the error code to the last network error that occurred. For more information about the errors that may occur, refer to the TelnetClient.ErrorCode enumeration.

See Also

TelnetClientException Class | SocketTools Namespace

TelnetClientException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]

```
Overrides Public ReadOnly Property Message As String
```

[C#]

```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

TelnetClientException Class | SocketTools Namespace

TelnetClientException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception. This value is the same as the values returned by the Windows Sockets API. For more information about socket error codes, see the Windows Socket Version 2 API error code documentation in MSDN.

See Also

TelnetClientException Class | SocketTools Namespace

TelnetClientException Methods

The methods of the **TelnetClientException** class are listed below. For a complete list of **TelnetClientException** class members, see the TelnetClientException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

TelnetClientException Class | SocketTools Namespace

TelnetClientException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]
```
Overrides Public Function ToString() As String
```

[C#]
```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

TelnetClientException Class | SocketTools Namespace

Terminal Class

Implements ANSI and DEC VT terminal emulation.

For a list of all members of this type, see Terminal Members.

System.Object
System.MarshalByRefObject
System.ComponentModel.Component
System.Windows.Forms.Control
System.Windows.Forms.ScrollableControl
System.Windows.Forms.ContainerControl
System.Windows.Forms.UserControl
SocketTools.Terminal

[Visual Basic]

```
<ToolboxBitmap, _  DefaultEvent(Name:="Load"), _  DesignerCategory(Category:="UserControl"), _
    Designer(DesignerBaseTypeName:="System.ComponentModel.Design.IDesigner",
   DesignerTypeName:="System.Windows.Forms.Design.ControlDesigner, System.Design,
   Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"), _
    Designer(DesignerBaseTypeName:="System.ComponentModel.Design.IRootDesigner, System,
   Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b77a5c561934e089",
   DesignerTypeName:="System.Windows.Forms.Design.UserControlDocumentDesigner, System.Design,
   Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"), _
    Designer(DesignerBaseTypeName:="System.ComponentModel.Design.IDesigner",
   DesignerTypeName:="System.Windows.Forms.Design.ScrollableControlDesigner, System.Design,
   Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"), _
    Designer(DesignerBaseTypeName:="System.ComponentModel.Design.IDesigner",
   DesignerTypeName:="System.Windows.Forms.Design.ControlDesigner, System.Design,
   Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"), _
    DefaultProperty(Name:="Text"), _  ToolboxItemFilter(FilterString:="System.Windows.Forms",
   FilterType:=ToolboxItemFilterType.Allow), _
    DesignerSerializer(SerializerTypeName:="System.Windows.Forms.Design.ControlCodeDomSerializer,
   System.Design, Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a",
   SerializerBaseTypeName:="System.ComponentModel.Design.Serialization.CodeDomSerializer,
   System.Design, Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a")>
Public Class Terminal
    Inherits UserControl
```

[C#]

```
[ToolboxBitmap]
[DefaultEvent(Name="Load")]
[DesignerCategory(Category="UserControl")]
[Designer(DesignerBaseTypeName="System.ComponentModel.Design.IDesigner",
   DesignerTypeName="System.Windows.Forms.Design.ControlDesigner, System.Design,
   Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a")]
[Designer(DesignerBaseTypeName="System.ComponentModel.Design.IRootDesigner, System,
   Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b77a5c561934e089",
   DesignerTypeName="System.Windows.Forms.Design.UserControlDocumentDesigner, System.Design,
   Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a")]
[Designer(DesignerBaseTypeName="System.ComponentModel.Design.IDesigner",
   DesignerTypeName="System.Windows.Forms.Design.ScrollableControlDesigner, System.Design,
   Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a")]
[Designer(DesignerBaseTypeName="System.ComponentModel.Design.IDesigner",
   DesignerTypeName="System.Windows.Forms.Design.ControlDesigner, System.Design,
   Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a")]
[DefaultProperty(Name="Text")]
[ToolboxItemFilter(FilterString="System.Windows.Forms",
   FilterType=ToolboxItemFilterType.Allow)]
[DesignerSerializer(SerializerTypeName="System.Windows.Forms.Design.ControlCodeDomSerializer,
   System.Design, Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a",
   SerializerBaseTypeName="System.ComponentModel.Design.Serialization.CodeDomSerializer,
   System.Design, Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a")]
public class Terminal : UserControl
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The Terminal control provides a comprehensive interface for emulating an ANSI, DEC VT100 or DEC VT220 character terminal, with full support for all standard escape and control sequences, color mapping and other advanced features. The control functions provide both a high level interface for parsing escape sequences and updating a display, as well as lower level primitives for directly managing the virtual display, such as controlling the individual display cells, moving the cursor position and specifying display attributes.

This control can be used in conjunction with the SocketTools.SshClient or SocketTools.TelnetClient classes to provide terminal emulation services for an application, or it can be used independently. For example, this control could also be used to provide emulation services for a program that requires serial modem connections to a server.

Requirements

Namespace: SocketTools

Assembly: SocketTools.Terminal (in SocketTools.Terminal.dll)

See Also

Terminal Members | SocketTools Namespace

Terminal Members

Terminal overview

Public Instance Constructors

Terminal Constructor	Initializes a new instance of the Terminal control.

Public Instance Fields

Cell	Gets and sets information about a character cell in the virtual display.
ColorMap	Gets and sets the color values used when displaying text.
KeyMap	Gets and sets the character sequence mapped to a special key.

Public Instance Properties

AccessibilityObject (inherited from Control)	Gets the AccessibleObject assigned to the control.
AccessibleDefaultActionDescription (inherited from Control)	Gets or sets the default action description of the control for use by accessibility client applications.
AccessibleDescription (inherited from Control)	Gets or sets the description of the control used by accessibility client applications.
AccessibleName (inherited from Control)	Gets or sets the name of the control used by accessibility client applications.
AccessibleRole (inherited from Control)	Gets or sets the accessible role of the control
ActiveControl (inherited from ContainerControl)	Gets or sets the active control on the container control.
AllowDrop (inherited from Control)	Gets or sets a value indicating whether the control can accept data that the user drags onto it.
Anchor (inherited from Control)	Gets or sets which edges of the control are anchored to the edges of its container.
Attributes	Gets and sets the current display attribute for the terminal emulator.
AutoRefresh	Enables and disables the automatic refreshing of the virtual display.
AutoScroll (inherited from ScrollableControl)	Gets or sets a value indicating whether the container will allow the user to scroll to any controls placed outside of its visible boundaries.
AutoScrollMargin (inherited from ScrollableControl)	Gets or sets the size of the auto-scroll margin.
AutoScrollMinSize (inherited from ScrollableControl)	Gets or sets the minimum size of the auto-scroll.
AutoScrollPosition (inherited from ScrollableControl)	Gets or sets the location of the auto-scroll position.

AutoSelect	Enables and disables the automatic selection of text in the display.
AutoSize	Enables and disables the automatic resizing of the display.
AutoWrap	Enables and disables the automatic wrapping of text in the virtual display.
BackColor (inherited from Control)	Gets or sets the background color for the control.
BackgroundImage (inherited from Control)	Gets or sets the background image displayed in the control.
Bell	Enables and disables the audible bell.
BindingContext (inherited from ContainerControl)	
BoldColor	Gets and sets the bold color of the control.
Bottom (inherited from Control)	Gets the distance between the bottom edge of the control and the top edge of its container's client area.
Bounds (inherited from Control)	Gets or sets the size and location of the control including its nonclient elements.
CanFocus (inherited from Control)	Gets a value indicating whether the control can receive focus.
CanSelect (inherited from Control)	Gets a value indicating whether the control can be selected.
Capture (inherited from Control)	Gets or sets a value indicating whether the control has captured the mouse.
Caret	Gets and sets the caret style of the control.
CausesValidation (inherited from Control)	Gets or sets a value indicating whether the control causes validation to be performed on any controls that require validation when it receives focus.
CellHeight	Gets the height of a character cell.
CellWidth	Gets the width of a character cell.
ClientRectangle (inherited from Control)	Gets the rectangle that represents the client area of the control.
ClientSize (inherited from Control)	Gets or sets the height and width of the client area of the control.
CodePage	Gets and sets the code page used when displaying text.
Columns	Gets and sets the number of columns in the virtual display.
CompanyName (inherited from Control)	Gets the name of the company or creator of the application containing the control.
Container (inherited from Component)	Gets the IContainer that contains the Component.

ContainsFocus (inherited from Control)	Gets a value indicating whether the control, or one of its child controls, currently has the input focus.
ContextMenu (inherited from Control)	Gets or sets the shortcut menu associated with the control.
Controls (inherited from Control)	Gets the collection of controls contained within the control.
Created (inherited from Control)	Gets a value indicating whether the control has been created.
Cursor (inherited from Control)	Gets or sets the cursor that is displayed when the mouse pointer is over the control.
CursorX	Gets and sets the current cursor position in the display.
CursorY	Gets and sets the current cursor position in the display.
DataBindings (inherited from Control)	Gets the data bindings for the control.
Display	Gets a value that specifies the virtual display handle allocated for the control.
DisplayRectangle (inherited from ScrollableControl)	
Disposing (inherited from Control)	Gets a value indicating whether the control is in the process of being disposed of.
Dock (inherited from Control)	Gets or sets which edge of the parent container a control is docked to.
DockPadding (inherited from ScrollableControl)	Gets the dock padding settings for all edges of the control.
Emulation	Gets and sets the emulation mode used by the control.
Enabled (inherited from Control)	Gets or sets a value indicating whether the control can respond to user interaction.
Focused (inherited from Control)	Gets a value indicating whether the control has input focus.
Font (inherited from Control)	Gets or sets the font of the text displayed by the control.
FontName	Gets and sets the name of the font used by the control.
FontSize	Gets and sets the size of the font used by the control.
ForeColor (inherited from Control)	Gets or sets the foreground color of the control.
Handle (inherited from Control)	Gets the window handle that the control is bound to.
HasChildren (inherited from Control)	Gets a value indicating whether the control contains one or more child controls.

Height (inherited from Control)	Gets or sets the height of the control.
ImeMode (inherited from Control)	Gets or sets the Input Method Editor (IME) mode of the control.
InvokeRequired (inherited from Control)	Gets a value indicating whether the caller must call an invoke method when making method calls to the control because the caller is on a different thread than the one the control was created on.
IsAccessible (inherited from Control)	Gets or sets a value indicating whether the control is visible to accessibility applications.
IsDisposed (inherited from Control)	Gets a value indicating whether the control has been disposed of.
IsHandleCreated (inherited from Control)	Gets a value indicating whether the control has a handle associated with it.
IsInitialized	Gets a value which indicates if the current instance of the control has been initialized successfully.
LastError	Gets or sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
Left (inherited from Control)	Gets or sets the x-coordinate of a control's left edge in pixels.
Location (inherited from Control)	Gets or sets the coordinates of the upper-left corner of the control relative to the upper-left corner of its container.
MouseX	Return the current mouse position in the display.
MouseY	Return the current mouse position in the display.
Name (inherited from Control)	Gets or sets the name of the control.
NewLine	Gets and sets the newline mode for the virtual display.
Parent (inherited from Control)	Gets or sets the parent container of the control.
ParentForm (inherited from ContainerControl)	Gets the form that the container control is assigned to.
ProductName (inherited from Control)	Gets the product name of the assembly containing the control.
ProductVersion (inherited from Control)	Gets the version of the assembly containing the control.
RecreatingHandle (inherited from Control)	Gets a value indicating whether the control is currently re-creating its handle.
Region (inherited from Control)	Gets or sets the window region associated with the control.
Right (inherited from Control)	Gets the distance between the right edge of the control and the left edge of its container.

RightToLeft (inherited from Control)	Gets or sets a value indicating whether control's elements are aligned to support locales using right-to-left fonts.
Rows	Gets and sets the number of rows in the virtual display.
ScrollBars	Gets and sets the scrollbar display mode for the control.
SelLength	Gets and sets the number of characters selected in the virtual display.
SelStart	Gets and sets the starting position of the current text selection.
SelText	Gets the selected text or text from a specific portion of the display.
Site (inherited from Control)	Gets or sets the site of the control.
Size (inherited from Control)	Gets or sets the height and width of the control.
TabIndex (inherited from Control)	Gets or sets the tab order of the control within its container.
TabStop (inherited from Control)	Gets or sets a value indicating whether the user can give the focus to this control using the TAB key.
Tag (inherited from Control)	Gets or sets the object that contains data about the control.
Text	Gets and sets the text displayed by the control.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Top (inherited from Control)	Gets or sets the y-coordinate of the control's top edge in pixels.
TopLevelControl (inherited from Control)	Gets the parent control that is not parented by another Windows Forms control. Typically, this is the outermost Form that the control is contained in.
Version	Gets a value which returns the current version of the Terminal control.
Visible (inherited from Control)	Gets or sets a value indicating whether the control is displayed.
Width (inherited from Control)	Gets or sets the width of the control.
WindowTarget (inherited from Control)	

Public Instance Methods

BeginInvoke (inherited from Control)	Overloaded. Executes the specified delegate asynchronously with the specified arguments, on the thread that the control's underlying handle

	was created on.
BringToFront (inherited from Control)	Brings the control to the front of the z-order.
Clear	Clear the terminal emulation window.
ClearEol	Erase all characters from the current column to the end of the line.
Contains (inherited from Control)	Retrieves a value indicating whether the specified control is a child of the control.
CreateControl (inherited from Control)	Forces the creation of the control, including the creation of the handle and any child controls.
CreateGraphics (inherited from Control)	Creates the Graphics object for the control.
CreateObjRef (inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
DelLine	Delete the current line in the terminal emulation display.
Deselect	Deselects any selected text in the display.
Dispose (inherited from Component)	Overloaded. Releases all resources used by the Component.
DoDragDrop (inherited from Control)	Begins a drag-and-drop operation.
EndInvoke (inherited from Control)	Retrieves the return value of the asynchronous operation represented by the IAsyncResult object passed.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
FindForm (inherited from Control)	Retrieves the form that the control is on.
Focus (inherited from Control)	Sets input focus to the control.
GetChildAtPoint (inherited from Control)	Retrieves the child control that is located at the specified coordinates.
GetContainerControl (inherited from Control)	Returns the next ContainerControl up the control's chain of parent controls.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService (inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetNextControl (inherited from Control)	Retrieves the next control forward or back in the tab order of child controls.
GetType (inherited from Object)	Gets the Type of the current instance.
Hide (inherited from Control)	Conceals the control from the user.
Initialize	Overloaded. Initialize an instance of the Terminal control.

InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
InsLine	Insert an empty line at the current position in the terminal emulation display.
Invalidate (inherited from Control)	Overloaded. Invalidates the specified region of the control (adds it to the control's update region, which is the area that will be repainted at the next paint operation), and causes a paint message to be sent to the control.
Invoke (inherited from Control)	Overloaded. Executes the specified delegate, on the thread that owns the control's underlying window handle, with the specified list of arguments.
PerformLayout (inherited from Control)	Overloaded. Forces the control to apply layout logic to all its child controls.
PointToClient (inherited from Control)	Computes the location of the specified screen point into client coordinates.
PointToScreen (inherited from Control)	Computes the location of the specified client point into screen coordinates.
PreProcessMessage	Preprocesses input messages within the message loop before they are dispatched.
RectangleToClient (inherited from Control)	Computes the size and location of the specified screen rectangle in client coordinates.
RectangleToScreen (inherited from Control)	Computes the size and location of the specified client rectangle in screen coordinates.
Refresh	Forces a complete redraw of the virtual display.
Reset	Reset the internal state of the control, resetting all properties to their default values.
ResetBackColor (inherited from Control)	Resets the BackColor property to its default value.
ResetBindings (inherited from Control)	Resets the DataBindings property to its default value.
ResetCursor (inherited from Control)	Resets the Cursor property to its default value.
ResetFont (inherited from Control)	Resets the Fontproperty to its default value.
ResetForeColor (inherited from Control)	Resets the ForeColor property to its default value.
ResetImeMode (inherited from Control)	Resets the ImeMode property to its default value.
ResetRightToLeft (inherited from Control)	Resets the RightToLeft property to its default value.
ResetText (inherited from Control)	Resets the Text property to its default value.
ResumeLayout (inherited from Control)	Overloaded. Resumes normal layout logic.
Scale (inherited from Control)	Overloaded. Scales the control and any child controls to the specified ratio.
ScrollControlIntoView (inherited from	

ScrollableControl)	
ScrollDown	Scroll the display down by one line.
ScrollUp	Scroll the display up by one line.
Select (inherited from Control)	Overloaded. Activates the control.
SelectNextControl (inherited from Control)	Activates the next control.
SelectText	Overloaded. Selects a region of the virtual display and returns the selected text.
SendToBack (inherited from Control)	Sends the control to the back of the z-order.
SetAutoScrollMargin (inherited from ScrollableControl)	Sets the size of the auto-scroll margins.
SetBounds (inherited from Control)	Overloaded. Sets the bounds of the control to the specified location and size.
Show (inherited from Control)	Displays the control to the user.
SuspendLayout (inherited from Control)	Temporarily suspends the layout logic for the control.
ToString (inherited from Component)	
Uninitialize	Uninitialize the control and release any resources allocated for the current process.
Update (inherited from Control)	Causes the control to redraw the invalidated regions within its client area.
Validate (inherited from ContainerControl)	Validates the last invalidated control and its ancestors up through, but not including, the current control.
Write	Overloaded. Write the contents of a byte array to the virtual display.

Public Instance Events

BackColorChanged (inherited from Control)	Occurs when the value of the BackColor property changes.
BackgroundImageChanged (inherited from Control)	Occurs when the value of the BackgroundImage property changes.
BindingContextChanged (inherited from Control)	Occurs when the value of the BindingContext property changes.
CausesValidationChanged (inherited from Control)	Occurs when the value of the CausesValidation property changes.
ChangeUICues (inherited from Control)	Occurs when the focus or keyboard user interface (UI) cues change.
Click (inherited from Control)	Occurs when the control is clicked.
ContextMenuChanged (inherited from Control)	Occurs when the value of the ContextMenu property changes.
ControlAdded (inherited from Control)	Occurs when a new control is added to the

	ControlCollection .
⚡ ControlRemoved (inherited from Control)	Occurs when a control is removed from the ControlCollection .
⚡ CursorChanged (inherited from Control)	Occurs when the value of the Cursor property changes.
⚡ Disposed (inherited from Component)	Adds an event handler to listen to the Disposed event on the component.
⚡ DockChanged (inherited from Control)	Occurs when the value of the Dock property changes.
⚡ DoubleClick (inherited from Control)	Occurs when the control is double-clicked.
⚡ DragDrop (inherited from Control)	Occurs when a drag-and-drop operation is completed.
⚡ DragEnter (inherited from Control)	Occurs when an object is dragged into the control's bounds.
⚡ DragLeave (inherited from Control)	Occurs when an object is dragged out of the control's bounds.
⚡ DragOver (inherited from Control)	Occurs when an object is dragged over the control's bounds.
⚡ EnabledChanged (inherited from Control)	Occurs when the Enabled property value has changed.
⚡ Enter (inherited from Control)	Occurs when the control is entered.
⚡ FontChanged (inherited from Control)	Occurs when the Font property value changes.
⚡ ForeColorChanged (inherited from Control)	Occurs when the ForeColor property value changes.
⚡ GiveFeedback (inherited from Control)	Occurs during a drag operation.
⚡ GotFocus (inherited from Control)	Occurs when the control receives focus.
⚡ HandleCreated (inherited from Control)	Occurs when a handle is created for the control.
⚡ HandleDestroyed (inherited from Control)	Occurs when the control's handle is in the process of being destroyed.
⚡ HelpRequested (inherited from Control)	Occurs when the user requests help for a control.
⚡ ImeModeChanged (inherited from Control)	Occurs when the ImeMode property has changed.
⚡ Invalidated (inherited from Control)	Occurs when a control's display requires redrawing.
⚡ KeyDown (inherited from Control)	Occurs when a key is pressed while the control has focus.
⚡ KeyMapped	Occurs when a mapped key is pressed.
⚡ KeyPress (inherited from Control)	Occurs when a key is pressed while the control has focus.
⚡ KeyUp (inherited from Control)	Occurs when a key is released while the control has focus.
⚡ Layout (inherited from Control)	Occurs when a control should reposition its child

	controls.
⚡ Leave (inherited from Control)	Occurs when the input focus leaves the control.
⚡ Load (inherited from UserControl)	Occurs before the control becomes visible for the first time.
⚡ LocationChanged (inherited from Control)	Occurs when the Location property value has changed.
⚡ LostFocus (inherited from Control)	Occurs when the control loses focus.
⚡ MouseDown (inherited from Control)	Occurs when the mouse pointer is over the control and a mouse button is pressed.
⚡ MouseEnter (inherited from Control)	Occurs when the mouse pointer enters the control.
⚡ MouseHover (inherited from Control)	Occurs when the mouse pointer hovers over the control.
⚡ MouseLeave (inherited from Control)	Occurs when the mouse pointer leaves the control.
⚡ MouseMove (inherited from Control)	Occurs when the mouse pointer is moved over the control.
⚡ MouseUp (inherited from Control)	Occurs when the mouse pointer is over the control and a mouse button is released.
⚡ MouseWheel (inherited from Control)	Occurs when the mouse wheel moves while the control has focus.
⚡ Move (inherited from Control)	Occurs when the control is moved.
⚡ Paint (inherited from Control)	Occurs when the control is redrawn.
⚡ ParentChanged (inherited from Control)	Occurs when the Parent property value changes.
⚡ QueryAccessibilityHelp (inherited from Control)	Occurs when AccessibleObject is providing help to accessibility applications.
⚡ QueryContinueDrag (inherited from Control)	Occurs during a drag-and-drop operation and allows the drag source to determine whether the drag-and-drop operation should be canceled.
⚡ Resize (inherited from Control)	Occurs when the control is resized.
⚡ RightToLeftChanged (inherited from Control)	Occurs when the RightToLeft property value changes.
⚡ SizeChanged (inherited from Control)	Occurs when the Size property value changes.
⚡ StyleChanged (inherited from Control)	Occurs when the control style changes.
⚡ SystemColorsChanged (inherited from Control)	Occurs when the system colors change.
⚡ TabIndexChanged (inherited from Control)	Occurs when the TabIndex property value changes.
⚡ TabStopChanged (inherited from Control)	Occurs when the TabStop property value changes.
⚡ TextChanged (inherited from UserControl)	
⚡ Validated (inherited from Control)	Occurs when the control is finished validating.
⚡ Validating (inherited from Control)	Occurs when the control is validating.
⚡	

VisibleChanged (inherited from Control)	Occurs when the Visible property value changes.

Protected Instance Properties

CreateParams (inherited from ContainerControl)	
DefaultImeMode (inherited from Control)	Gets the default Input Method Editor (IME) mode supported by the control.
DefaultSize (inherited from UserControl)	
DesignMode (inherited from Component)	Gets a value that indicates whether the Component is currently in design mode.
Events (inherited from Component)	Gets the list of event handlers that are attached to this Component.
FontHeight (inherited from Control)	Gets or sets the height of the font of the control.
HScroll (inherited from ScrollableControl)	Gets or sets a value indicating whether the horizontal scroll bar is visible.
RenderRightToLeft (inherited from Control)	
ResizeRedraw (inherited from Control)	Gets or sets a value indicating whether the control redraws itself when resized.
ShowFocusCues (inherited from Control)	Gets a value indicating whether the control should display focus rectangles.
ShowKeyboardCues (inherited from Control)	Gets a value indicating whether the control should display keyboard shortcuts.
VScroll (inherited from ScrollableControl)	Gets or sets a value indicating whether the vertical scroll bar is visible.

Protected Instance Methods

AccessibilityNotifyClients (inherited from Control)	Notifies the accessibility client applications of the specified AccessibleEvents for the specified child control.
AdjustFormScrollbars (inherited from ContainerControl)	
CreateAccessibilityInstance (inherited from Control)	Creates a new accessibility object for the control.
CreateControlsInstance (inherited from Control)	Creates a new instance of the control collection for the control.
CreateHandle (inherited from Control)	Creates a handle for the control.
DefWndProc (inherited from Control)	Sends the specified message to the default window procedure.
DestroyHandle (inherited from Control)	Destroys the handle associated with the control.
Dispose	Overloaded. Releases the unmanaged resources allocated by the Terminal class and optionally releases the managed resources.
Finalize	Destroys an instance of the control, releasing the

	resources allocated for the virtual display.
GetScrollState (inherited from ScrollableControl)	
GetService (inherited from Component)	Returns an object that represents a service provided by the Component or by its Container.
GetStyle (inherited from Control)	Retrieves the value of the specified control style bit for the control.
GetTopLevel (inherited from Control)	Determines if the control is a top-level control.
InitLayout (inherited from Control)	Called after the control has been added to another container.
InvokeGotFocus (inherited from Control)	Raises the GotFocus event for the specified control.
InvokeLostFocus (inherited from Control)	Raises the LostFocus event for the specified control.
InvokeOnClick (inherited from Control)	Raises the Click event for the specified control.
InvokePaint (inherited from Control)	Raises the Paint event for the specified control.
InvokePaintBackground (inherited from Control)	Raises the PaintBackground event for the specified control.
IsInputChar (inherited from Control)	Determines if a character is an input character that the control recognizes.
IsInputKey (inherited from Control)	Determines whether the specified key is a regular input key or a special key that requires preprocessing.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.
NotifyInvalidate (inherited from Control)	
OnBackColorChanged	Raises the BackColorChanged event.
OnBackgroundImageChanged (inherited from Control)	Raises the BackgroundImageChanged event.
OnBindingContextChanged (inherited from Control)	Raises the BindingContextChangedevent.
OnCausesValidationChanged (inherited from Control)	Raises the CausesValidationChangedevent.
OnChangeUICues (inherited from Control)	Raises the ChangeUICues event.
OnClick (inherited from Control)	Raises the Click event.
OnContextMenuChanged (inherited from Control)	Raises the ContextMenuChangedevent.
OnControlAdded (inherited from Control)	Raises the ControlAdded event.
OnControlRemoved (inherited from ContainerControl)	
OnCreateControl (inherited from UserControl)	
OnCursorChanged (inherited from Control)	Raises the CursorChangedevent.

OnDockChanged (inherited from Control)	Raises the DockChangedevent.
OnDoubleClick (inherited from Control)	Raises the DoubleClick event.
OnDragDrop (inherited from Control)	Raises the DragDrop event.
OnDragEnter (inherited from Control)	Raises the DragEnter event.
OnDragLeave (inherited from Control)	Raises the DragLeave event.
OnDragOver (inherited from Control)	Raises the DragOver event.
OnEnabledChanged (inherited from Control)	Raises the EnabledChanged event.
OnEnter (inherited from Control)	Raises the Enter event.
OnFontChanged	Raises the FontChanged event.
OnForeColorChanged	Raises the ForeColorChanged event.
OnGiveFeedback (inherited from Control)	Raises the GiveFeedback event.
OnGotFocus	Raises the GotFocus event.
OnHandleCreated (inherited from Control)	Raises the HandleCreated event.
OnHandleDestroyed (inherited from Control)	Raises the HandleDestroyed event.
OnHelpRequested (inherited from Control)	Raises the HelpRequested event.
OnImeModeChanged (inherited from Control)	Raises the ImeModeChanged event.
OnInvalidated (inherited from Control)	Raises the Invalidated event.
OnKeyDown (inherited from Control)	Raises the KeyDown event.
OnKeyPress (inherited from Control)	Raises the KeyPress event.
OnKeyUp (inherited from Control)	Raises the KeyUp event.
OnLayout (inherited from ScrollableControl)	
OnLeave (inherited from Control)	Raises the Leave event.
OnLoad (inherited from UserControl)	Raises the Load event.
OnLocationChanged (inherited from Control)	Raises the LocationChanged event.
OnLostFocus	Raises the LostFocus event.
OnMouseDown	Raises the MouseDown event.
OnMouseEnter (inherited from Control)	Raises the MouseEnter event.
OnMouseHover (inherited from Control)	Raises the MouseHover event.
OnMouseLeave (inherited from Control)	Raises the MouseLeave event.
OnMouseMove	Raises the MouseMove event.
OnMouseUp	Raises the MouseUp event.
OnMouseWheel (inherited from ScrollableControl)	
OnMove (inherited from Control)	Raises the Move event.
OnNotifyMessage (inherited from Control)	Notifies the control of Windows messages.
OnPaint	Raises the Paint event.

OnPaintBackground (inherited from Control)	Paints the background of the control.
OnParentBackColorChanged (inherited from Control)	Raises the BackColorChanged event when the BackColor property value of the control's container changes.
OnParentBackgroundImageChanged (inherited from Control)	Raises the BackgroundImageChanged event when the BackgroundImage property value of the control's container changes.
OnParentBindingContextChanged (inherited from Control)	Raises the BindingContextChanged event when the BindingContext property value of the control's container changes.
OnParentChanged	Raises the ParentChanged event.
OnParentEnabledChanged (inherited from Control)	Raises the EnabledChanged event when the Enabled property value of the control's container changes.
OnParentFontChanged (inherited from Control)	Raises the FontChanged event when the Font property value of the control's container changes.
OnParentForeColorChanged (inherited from Control)	Raises the ForeColorChanged event when the ForeColor property value of the control's container changes.
OnParentRightToLeftChanged (inherited from Control)	Raises the RightToLeftChanged event when the RightToLeft property value of the control's container changes.
OnParentVisibleChanged (inherited from Control)	Raises the VisibleChanged event when the Visible property value of the control's container changes.
OnQueryContinueDrag (inherited from Control)	Raises the QueryContinueDrag event.
OnResize	Raises the Resize event.
OnRightToLeftChanged (inherited from Control)	Raises the RightToLeftChangedevent.
OnSizeChanged (inherited from Control)	Raises the SizeChanged event.
OnStyleChanged (inherited from Control)	Raises the StyleChanged event.
OnSystemColorsChanged (inherited from Control)	Raises the SystemColorsChanged event.
OnTabIndexChanged (inherited from Control)	Raises the TabIndexChanged event.
OnTabStopChanged (inherited from Control)	Raises the TabStopChanged event.
OnTextChanged (inherited from Control)	Raises the TextChanged event.
OnValidated (inherited from Control)	Raises the Validated event.
OnValidating (inherited from Control)	Raises the Validating event.
OnVisibleChanged (inherited from ScrollableControl)	
ProcessCmdKey (inherited from Control)	Processes a command key.
ProcessDialogChar (inherited from ContainerControl)	

ProcessDialogKey (inherited from ContainerControl)	
ProcessKeyEventArgs (inherited from Control)	Processes a key message and generates the appropriate control events.
ProcessKeyPreview (inherited from Control)	Previews a keyboard message.
ProcessMnemonic (inherited from ContainerControl)	
ProcessTabKey (inherited from ContainerControl)	Selects the next available control and makes it the active control.
RaiseDragEvent (inherited from Control)	
RaiseKeyEvent (inherited from Control)	
RaiseMouseEvent (inherited from Control)	
RaisePaintEvent (inherited from Control)	
RecreateHandle (inherited from Control)	Forces the re-creation of the handle for the control.
ResetMouseEventArgs (inherited from Control)	
RtlTranslateAlignment (inherited from Control)	Overloaded. Converts the specified HorizontalAlignment to the appropriate HorizontalAlignment to support right-to-left text.
RtlTranslateContent (inherited from Control)	Converts the specified ContentAlignment to the appropriate ContentAlignment to support right-to-left text.
RtlTranslateHorizontal (inherited from Control)	Converts the specified HorizontalAlignment to the appropriate HorizontalAlignment to support right-to-left text.
RtlTranslateLeftRight (inherited from Control)	Converts the specified LeftRightAlignment to the appropriate LeftRightAlignment to support right-to-left text.
ScaleCore (inherited from ScrollableControl)	
Select (inherited from ContainerControl)	Overloaded.
SetBoundsCore (inherited from Control)	Performs the work of setting the specified bounds of this control.
SetClientSizeCore (inherited from Control)	Sets the size of the client area of the control.
SetDisplayRectLocation (inherited from ScrollableControl)	
SetScrollState (inherited from ScrollableControl)	
SetStyle (inherited from Control)	Sets the specified style bit to the specified value.
SetTopLevel (inherited from Control)	Sets the control as the top-level control.
SetVisibleCore (inherited from Control)	Sets the control to the specified visible state.
UpdateBounds (inherited from Control)	Overloaded. Updates the bounds of the control with the current size and location.

UpdateDefaultButton (inherited from ContainerControl)	
UpdateStyles (inherited from Control)	Forces the assigned styles to be reapplied to the control.
UpdateZOrder (inherited from Control)	Updates the control in its parent's z-order.
WndProc	Processes Windows messages.

Protected Internal Instance Methods

ProcessKeyMessage (inherited from Control)	Processes a keyboard message.

See Also

Terminal Class | SocketTools Namespace

Terminal Constructor

Initializes a new instance of the Terminal control.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public Terminal();
```

See Also

Terminal Class | SocketTools Namespace

Terminal.TerminalCell Structure

This structure is used by the Cell property to return information about a specific character cell in the virtual display.

For a list of all members of this type, see Terminal.TerminalCell Members.

System.Object
  System.ValueType
    SocketTools.Terminal.TerminalCell

[Visual Basic]
```
Public Structure Terminal.TerminalCell
```

[C#]
```
public struct Terminal.TerminalCell
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: SocketTools

Assembly: SocketTools.Terminal (in SocketTools.Terminal.dll)

See Also

Terminal.TerminalCell Members | SocketTools Namespace

Terminal.TerminalCell Members

Terminal.TerminalCell overview

Public Instance Constructors

Terminal.TerminalCell	Overloaded. Initializes a new instance of the Terminal.TerminalCell class.

Public Instance Fields

Attributes	Gets the character attributes.
Character	Gets the character value.

Public Instance Methods

Equals (inherited from ValueType)	Indicates whether this instance and a specified object are equal.
GetHashCode (inherited from ValueType)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from ValueType)	Returns the fully qualified type name of this instance.

See Also

Terminal.TerminalCell Class | SocketTools Namespace

Terminal.TerminalCell Constructor

Initializes a new instance of the TerminalCell structure with the specified character.

Overload List

Initializes a new instance of the TerminalCell structure with the specified character.

public Terminal.TerminalCell(char);

Initializes a new instance of the TerminalCell structure with the specified character and attributes.

public Terminal.TerminalCell(char,TerminalAttributes);

See Also

Terminal.TerminalCell Class | SocketTools Namespace

Terminal.TerminalCell Constructor (Char)

Initializes a new instance of the TerminalCell structure with the specified character.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal character As Char _
)
```

[C#]

```
public Terminal.TerminalCell(
   char character
);
```

Parameters

character

The character value used to initialize the structure.

Remarks

The **TerminalCell** structure will be initialized with the specified character. The default, normal attributes for the character cell will be used.

See Also

Terminal.TerminalCell Class | SocketTools Namespace | Terminal.TerminalCell Constructor Overload List

Terminal.TerminalCell Constructor (Char, TerminalAttributes)

Initializes a new instance of the TerminalCell structure with the specified character and attributes.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal character As Char, _
   ByVal attributes As TerminalAttributes _
)
```

[C#]

```
public Terminal.TerminalCell(
   char character,
   TerminalAttributes attributes
);
```

Parameters

character

The character value used to initialize the structure.

attributes

A TerminalAttributes enumeration value which specifies the attributes for the character cell.

Remarks

The **TerminalCell** structure will be initialized with the specified character and attributes.

See Also

Terminal.TerminalCell Class | SocketTools Namespace | Terminal.TerminalCell Constructor Overload List

Terminal.TerminalCell Fields

The fields of the **Terminal.TerminalCell** structure are listed below. For a complete list of **Terminal.TerminalCell** structure members, see the Terminal.TerminalCell Members topic.

Public Instance Fields

◆ Attributes	Gets the character attributes.
◆ Character	Gets the character value.

See Also

Terminal.TerminalCell Class | SocketTools Namespace

Terminal.TerminalCell.Attributes Field

Gets the character attributes.

[Visual Basic]

```
Public Attributes As TerminalAttributes
```

[C#]

```
public TerminalAttributes Attributes;
```

See Also

Terminal.TerminalCell Class | SocketTools Namespace

Terminal.TerminalCell.Character Field

Gets the character value.

[Visual Basic]

```
Public Character As Char
```

[C#]

```
public char Character;
```

See Also

Terminal.TerminalCell Class | SocketTools Namespace

Terminal Properties

The properties of the **Terminal** class are listed below. For a complete list of **Terminal** class members, see the Terminal Members topic.

Public Instance Properties

AccessibilityObject (inherited from Control)	Gets the AccessibleObject assigned to the control.
AccessibleDefaultActionDescription (inherited from Control)	Gets or sets the default action description of the control for use by accessibility client applications.
AccessibleDescription (inherited from Control)	Gets or sets the description of the control used by accessibility client applications.
AccessibleName (inherited from Control)	Gets or sets the name of the control used by accessibility client applications.
AccessibleRole (inherited from Control)	Gets or sets the accessible role of the control
ActiveControl (inherited from ContainerControl)	Gets or sets the active control on the container control.
AllowDrop (inherited from Control)	Gets or sets a value indicating whether the control can accept data that the user drags onto it.
Anchor (inherited from Control)	Gets or sets which edges of the control are anchored to the edges of its container.
Attributes	Gets and sets the current display attribute for the terminal emulator.
AutoRefresh	Enables and disables the automatic refreshing of the virtual display.
AutoScroll (inherited from ScrollableControl)	Gets or sets a value indicating whether the container will allow the user to scroll to any controls placed outside of its visible boundaries.
AutoScrollMargin (inherited from ScrollableControl)	Gets or sets the size of the auto-scroll margin.
AutoScrollMinSize (inherited from ScrollableControl)	Gets or sets the minimum size of the auto-scroll.
AutoScrollPosition (inherited from ScrollableControl)	Gets or sets the location of the auto-scroll position.
AutoSelect	Enables and disables the automatic selection of text in the display.
AutoSize	Enables and disables the automatic resizing of the display.
AutoWrap	Enables and disables the automatic wrapping of text in the virtual display.
BackColor (inherited from Control)	Gets or sets the background color for the control.
BackgroundImage (inherited from Control)	Gets or sets the background image displayed in the control.

Bell	Enables and disables the audible bell.
BindingContext (inherited from ContainerControl)	
BoldColor	Gets and sets the bold color of the control.
Bottom (inherited from Control)	Gets the distance between the bottom edge of the control and the top edge of its container's client area.
Bounds (inherited from Control)	Gets or sets the size and location of the control including its nonclient elements.
CanFocus (inherited from Control)	Gets a value indicating whether the control can receive focus.
CanSelect (inherited from Control)	Gets a value indicating whether the control can be selected.
Capture (inherited from Control)	Gets or sets a value indicating whether the control has captured the mouse.
Caret	Gets and sets the caret style of the control.
CausesValidation (inherited from Control)	Gets or sets a value indicating whether the control causes validation to be performed on any controls that require validation when it receives focus.
CellHeight	Gets the height of a character cell.
CellWidth	Gets the width of a character cell.
ClientRectangle (inherited from Control)	Gets the rectangle that represents the client area of the control.
ClientSize (inherited from Control)	Gets or sets the height and width of the client area of the control.
CodePage	Gets and sets the code page used when displaying text.
Columns	Gets and sets the number of columns in the virtual display.
CompanyName (inherited from Control)	Gets the name of the company or creator of the application containing the control.
Container (inherited from Component)	Gets the IContainer that contains the Component.
ContainsFocus (inherited from Control)	Gets a value indicating whether the control, or one of its child controls, currently has the input focus.
ContextMenu (inherited from Control)	Gets or sets the shortcut menu associated with the control.
Controls (inherited from Control)	Gets the collection of controls contained within the control.
Created (inherited from Control)	Gets a value indicating whether the control has been created.
Cursor (inherited from Control)	Gets or sets the cursor that is displayed when the mouse pointer is over the control.

CursorX	Gets and sets the current cursor position in the display.
CursorY	Gets and sets the current cursor position in the display.
DataBindings (inherited from Control)	Gets the data bindings for the control.
Display	Gets a value that specifies the virtual display handle allocated for the control.
DisplayRectangle (inherited from ScrollableControl)	
Disposing (inherited from Control)	Gets a value indicating whether the control is in the process of being disposed of.
Dock (inherited from Control)	Gets or sets which edge of the parent container a control is docked to.
DockPadding (inherited from ScrollableControl)	Gets the dock padding settings for all edges of the control.
Emulation	Gets and sets the emulation mode used by the control.
Enabled (inherited from Control)	Gets or sets a value indicating whether the control can respond to user interaction.
Focused (inherited from Control)	Gets a value indicating whether the control has input focus.
Font (inherited from Control)	Gets or sets the font of the text displayed by the control.
FontName	Gets and sets the name of the font used by the control.
FontSize	Gets and sets the size of the font used by the control.
ForeColor (inherited from Control)	Gets or sets the foreground color of the control.
Handle (inherited from Control)	Gets the window handle that the control is bound to.
HasChildren (inherited from Control)	Gets a value indicating whether the control contains one or more child controls.
Height (inherited from Control)	Gets or sets the height of the control.
ImeMode (inherited from Control)	Gets or sets the Input Method Editor (IME) mode of the control.
InvokeRequired (inherited from Control)	Gets a value indicating whether the caller must call an invoke method when making method calls to the control because the caller is on a different thread than the one the control was created on.
IsAccessible (inherited from Control)	Gets or sets a value indicating whether the control is visible to accessibility applications.
IsDisposed (inherited from Control)	Gets a value indicating whether the control has

	been disposed of.
IsHandleCreated (inherited from Control)	Gets a value indicating whether the control has a handle associated with it.
IsInitialized	Gets a value which indicates if the current instance of the control has been initialized successfully.
LastError	Gets or sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
Left (inherited from Control)	Gets or sets the x-coordinate of a control's left edge in pixels.
Location (inherited from Control)	Gets or sets the coordinates of the upper-left corner of the control relative to the upper-left corner of its container.
MouseX	Return the current mouse position in the display.
MouseY	Return the current mouse position in the display.
Name (inherited from Control)	Gets or sets the name of the control.
NewLine	Gets and sets the newline mode for the virtual display.
Parent (inherited from Control)	Gets or sets the parent container of the control.
ParentForm (inherited from ContainerControl)	Gets the form that the container control is assigned to.
ProductName (inherited from Control)	Gets the product name of the assembly containing the control.
ProductVersion (inherited from Control)	Gets the version of the assembly containing the control.
RecreatingHandle (inherited from Control)	Gets a value indicating whether the control is currently re-creating its handle.
Region (inherited from Control)	Gets or sets the window region associated with the control.
Right (inherited from Control)	Gets the distance between the right edge of the control and the left edge of its container.
RightToLeft (inherited from Control)	Gets or sets a value indicating whether control's elements are aligned to support locales using right-to-left fonts.
Rows	Gets and sets the number of rows in the virtual display.
ScrollBars	Gets and sets the scrollbar display mode for the control.
SelLength	Gets and sets the number of characters selected in the virtual display.
SelStart	Gets and sets the starting position of the current

	text selection.
SelText	Gets the selected text or text from a specific portion of the display.
Site (inherited from Control)	Gets or sets the site of the control.
Size (inherited from Control)	Gets or sets the height and width of the control.
TabIndex (inherited from Control)	Gets or sets the tab order of the control within its container.
TabStop (inherited from Control)	Gets or sets a value indicating whether the user can give the focus to this control using the TAB key.
Tag (inherited from Control)	Gets or sets the object that contains data about the control.
Text	Gets and sets the text displayed by the control.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Top (inherited from Control)	Gets or sets the y-coordinate of the control's top edge in pixels.
TopLevelControl (inherited from Control)	Gets the parent control that is not parented by another Windows Forms control. Typically, this is the outermost Form that the control is contained in.
Version	Gets a value which returns the current version of the Terminal control.
Visible (inherited from Control)	Gets or sets a value indicating whether the control is displayed.
Width (inherited from Control)	Gets or sets the width of the control.
WindowTarget (inherited from Control)	

Protected Instance Properties

CreateParams (inherited from ContainerControl)	
DefaultImeMode (inherited from Control)	Gets the default Input Method Editor (IME) mode supported by the control.
DefaultSize (inherited from UserControl)	
DesignMode (inherited from Component)	Gets a value that indicates whether the Component is currently in design mode.
Events (inherited from Component)	Gets the list of event handlers that are attached to this Component.
FontHeight (inherited from Control)	Gets or sets the height of the font of the control.
HScroll (inherited from ScrollableControl)	Gets or sets a value indicating whether the horizontal scroll bar is visible.
RenderRightToLeft (inherited from Control)	

ResizeRedraw (inherited from Control)	Gets or sets a value indicating whether the control redraws itself when resized.
ShowFocusCues (inherited from Control)	Gets a value indicating whether the control should display focus rectangles.
ShowKeyboardCues (inherited from Control)	Gets a value indicating whether the control should display keyboard shortcuts.
VScroll (inherited from ScrollableControl)	Gets or sets a value indicating whether the vertical scroll bar is visible.

See Also

Terminal Class | SocketTools Namespace

Terminal.Attributes Property

Gets and sets the current display attribute for the terminal emulator.

[Visual Basic]

```
<Description(Description:="Set or return the current display attribute for the
   terminal emulator."), _  Browsable(Browsable:=False)>
Public Property Attributes As TerminalAttributes
```

[C#]

```
[Description(Description="Set or return the current display attribute for the
   terminal emulator.")]
[Browsable(Browsable=False)]
public Terminal.TerminalAttributes Attributes {get; set;}
```

Property Value

A TerminalAttributes enumeration value which specifies one or more display attributes.

Remarks

The **Attributes** property can be used to determine the current display attributes, or to change the current attribute for subsequent text.

See Also

Terminal Class | SocketTools Namespace

Terminal.AutoRefresh Property

Enables and disables the automatic refreshing of the virtual display.

[Visual Basic]

```
<Category(Category:="Design"), _  Description(Description:="Enable or disable the
   automatic refreshing of the virtual display.")>
Public Property AutoRefresh As Boolean
```

[C#]

```
[Category(Category="Design")]
[Description(Description="Enable or disable the automatic refreshing of the virtual
   display.")]
public bool AutoRefresh {get; set;}
```

Property Value

A boolean value which specifies if the display is automatically refreshed.

Remarks

The **AutoRefresh** property is used to enable or disable the automatic refreshing of the virtual display whenever characters are written or the cursor position changes. By setting the property to **false**, the display can be changed and those changes will not be displayed until the property is reset to **true**. This allows an application to make a series of changes to the display text, attributes or cursor position without causing it to flicker.

See Also

Terminal Class | SocketTools Namespace

Terminal.AutoSelect Property

Enables and disables the automatic selection of text in the display.

[Visual Basic]

```
<Description(Description:="Enable or disable the automatic selection of text in the
   display."), _  Category(Category:="Design")>
Public Property AutoSelect As Boolean
```

[C#]

```
[Description(Description="Enable or disable the automatic selection of text in the
   display.")]
[Category(Category="Design")]
public bool AutoSelect {get; set;}
```

Property Value

A boolean value which specifies if text is automatically selected using the mouse.

Remarks

The **AutoSelect** property is used to enable or disable the automatic selection of text in the virtual display. When the property is set to **true**, the user can select text by clicking and dragging the mouse over the text to be selected. When set to **false**, no text is selected if the user drags the mouse over the display.

See Also

Terminal Class | SocketTools Namespace

Terminal.AutoSize Property

Enables and disables the automatic resizing of the display.

[Visual Basic]
```
<Category(Category:="Design"), _  Description(Description:="Enable or disable the
   automatic sizing of the display.")>
Public Property AutoSize As Boolean
```

[C#]
```
[Category(Category="Design")]
[Description(Description="Enable or disable the automatic sizing of the display.")]
public bool AutoSize {get; set;}
```

Property Value

A boolean value which specifies if the display is automatically resized.

Remarks

The **AutoSize** property determines if the control window is automatically resized when the parent window changes size. A value of **true** specifies that the control window should automatically resize itself to the size of its parent window. A value of **false** specifies that the control window should remain unchanged when the parent window is resized.

See Also

Terminal Class | SocketTools Namespace

Terminal.AutoWrap Property

Enables and disables the automatic wrapping of text in the virtual display.

[Visual Basic]

```
<Description(Description:="Enable or disable the wrapping of text in the display."),
   _  Category(Category:="Design")>
Public Property AutoWrap As Boolean
```

[C#]

```
[Description(Description="Enable or disable the wrapping of text in the display.")]
[Category(Category="Design")]
public bool AutoWrap {get; set;}
```

Property Value

A boolean value which specifies if automatic line wrapping is enabled.

Remarks

The **AutoWrap** property enables or disables the wrapping of text in the virtual display. If set to **true**, the cursor is re-positioned to the first column on the next line when text reaches last column in the display. If set to **false**, any text displayed beyond the last column is discarded.

See Also

Terminal Class | SocketTools Namespace

Terminal.Bell Property

Enables and disables the audible bell.

[Visual Basic]

```
<Category(Category:="Design"), _  Description(Description:="Enable or disable the
   audible bell.")>
Public Property Bell As Boolean
```

[C#]

```
[Category(Category="Design")]
[Description(Description="Enable or disable the audible bell.")]
public bool Bell {get; set;}
```

Property Value

A boolean value which specifies if the audible bell is enabled.

Remarks

The **Bell** property enables or disables the audible bell which is played whenever the control character Ctrl+G is encountered. The default property value is **true**.

See Also

Terminal Class | SocketTools Namespace

Terminal.BoldColor Property

Gets and sets the bold color of the control.

[Visual Basic]

```
<Category(Category:="Appearance"), _  Description(Description:="Sets or returns the
   bold color for the control.")>
Public Property BoldColor As Color
```

[C#]

```
[Category(Category="Appearance")]
[Description(Description="Sets or returns the bold color for the control.")]
public System.Drawing.Color BoldColor {get; set;}
```

Property Value

A **Color** value which specifies the text foreground color to use when the bold attribute is set.

Remarks

The **BoldColor** property returns the current color used for bold text in the control. Setting the property changes the color to the specified value.

See Also

Terminal Class | SocketTools Namespace

Terminal.Caret Property

Gets and sets the caret style of the control.

[Visual Basic]

```
<Category(Category:="Design"), _  Description(Description:="Set or return the style
   of caret displayed in the emulator.")>
Public Property Caret As TerminalCaret
```

[C#]

```
[Category(Category="Design")]
[Description(Description="Set or return the style of caret displayed in the
   emulator.")]
public Terminal.TerminalCaret Caret {get; set;}
```

Property Value

A TerminalCaret enumeration value which specifies the caret style.

See Also

Terminal Class | SocketTools Namespace

Terminal.Cell Field

Gets and sets information about a character cell in the virtual display.

[Visual Basic]

```
<Browsable(Browsable:=False), _  Description(Description:="Returns information about
   the specified character cell in the display.")>
Public ReadOnly Cell As DisplayCells
```

[C#]

```
[Browsable(Browsable=False)]
[Description(Description="Returns information about the specified character cell in
   the display.")]
public readonly DisplayCells Cell;
```

Remarks

The **Cell** property is used to determine the character and attributes for a specific character cell in the virtual display. The contents of the character cell can also be changed by assigning a value to this property.

See Also

Terminal Class | SocketTools Namespace

Terminal.CellHeight Property

Gets the height of a character cell.

[Visual Basic]

```
<Category(Category:="Layout"), _  ReadOnly(IsReadOnly:=True), _
    Description(Description:="Returns the height of a text cell in pixels.")>
Public ReadOnly Property CellHeight As Integer
```

[C#]

```
[Category(Category="Layout")]
[ReadOnly(IsReadOnly=True)]
[Description(Description="Returns the height of a text cell in pixels.")]
public int CellHeight {get;}
```

Property Value

An integer value which specifies the height of a character cell in pixels.

Remarks

This value can be used in calculating the minimum height of the control window in order to display all lines of text.

See Also

Terminal Class | SocketTools Namespace

Terminal.CellWidth Property

Gets the width of a character cell.

[Visual Basic]

```
<Description(Description:="Returns the width of a text cell in pixels."), _
    Category(Category:="Layout"), _  ReadOnly(IsReadOnly:=True)>
Public ReadOnly Property CellWidth As Integer
```

[C#]

```
[Description(Description="Returns the width of a text cell in pixels.")]
[Category(Category="Layout")]
[ReadOnly(IsReadOnly=True)]
public int CellWidth {get;}
```

Property Value

An integer value which specifies the width of a character cell in pixels.

Remarks

This value can be used in calculating the minimum width of the control window in order to display all lines of text.

See Also

Terminal Class | SocketTools Namespace

Terminal.CodePage Property

Gets and sets the code page used when displaying text.

[Visual Basic]

```
Public Property CodePage As Integer
```

[C#]

```
public int CodePage {get; set;}
```

Property Value

An integer value which specifies the current code page. A value of zero specifies the default code page for the current locale should be used. To preserve the original Unicode text, you can use code page 65001 which specifies UTF-8 character encoding.

Remarks

By default, strings are converted to an array of bytes using the code page for the current locale, mapping the 16-bit Unicode characters to bytes. Similarly, when reading text from the virtual display into a string buffer, the characters are converted to Unicode before they are returned to your application.

If the text appears to corrupted or characters are being replaced with question marks or other symbols, it is likely the text is using a different character encoding. Most services use UTF-8 encoding to represent non-ASCII characters and selecting the UTF-8 code page will typically resolve the issue.

For backwards compatibility, this class defaults to using the code page for the current locale. This property value directly corresponds to Windows code page identifiers, and will accept any valid code page supported by the .NET Framework. Setting this property to an invalid code page will generate an exception.

See Also

Terminal Class | SocketTools Namespace

Terminal.ColorMap Field

Gets and sets the color values used when displaying text.

[Visual Basic]

```
<Browsable(Browsable:=False), _  Description(Description:="Sets or returns the color
   value used when displaying color text attributes.")>
Public ReadOnly ColorMap As DisplayColorMap
```

[C#]

```
[Browsable(Browsable=False)]
[Description(Description="Sets or returns the color value used when displaying color
   text attributes.")]
public readonly DisplayColorMap ColorMap;
```

Remarks

The **ColorMap** array provides access to the virtual display color table which determines what color values are used to display foreground and background text color attributes.

When the emulator processes an escape sequence that changes the current foreground or background color, the actual RGB color value is determined by looking up the value in the virtual display's color table. The **ColorMap** property is useful for determining what values are being used when a color attribute is set and enables an application to change those colors. The emulator currently supports a maximum of sixteen (16) color values, and the index into the table corresponds to the color as defined by the standard for ANSI terminals.

See Also

Terminal Class | SocketTools Namespace

Terminal.Columns Property

Gets and sets the number of columns in the virtual display.

[Visual Basic]

```
<Description(Description:="Set or return the number of columns in the emulation
   display."), _  Category(Category:="Layout")>
Public Property Columns As Integer
```

[C#]

```
[Description(Description="Set or return the number of columns in the emulation
   display.")]
[Category(Category="Layout")]
public int Columns {get; set;}
```

Property Value

An integer value which specifies the number of columns.

Remarks

The **Columns** property returns the number of columns in the virtual display, or allows the application to change the number of columns. Currently, the number of columns may only be set to 80 or 132. Note that changing the number of columns in the display causes the current display to be invalidated, and the window will be cleared.

See Also

Terminal Class | SocketTools Namespace

Terminal.CursorX Property

Gets and sets the current cursor position in the display.

[Visual Basic]

```
<Description(Description:="Set or return the current column in the display."), _
    Browsable(Browsable:=False)>
Public Property CursorX As Integer
```

[C#]

```
[Description(Description="Set or return the current column in the display.")]
[Browsable(Browsable=False)]
public int CursorX {get; set;}
```

Property Value

An integer value which specifies the current cursor column.

Remarks

The **CursorX** property returns the current position of the cursor in the display, or can be used to change the current position. The current position is given in columns and indicates where the next text character will be displayed. To calculate the pixel offset where the cursor is located in the control window, multiply this value by the **CellWidth** property value.

See Also

Terminal Class | SocketTools Namespace

Terminal.CursorY Property

Gets and sets the current cursor position in the display.

[Visual Basic]

```
<Browsable(Browsable:=False), _  Description(Description:="Set or return the current
   row in the display.")>
Public Property CursorY As Integer
```

[C#]

```
[Browsable(Browsable=False)]
[Description(Description="Set or return the current row in the display.")]
public int CursorY {get; set;}
```

Property Value

An integer value which specifies the current cursor row.

Remarks

The **CursorY** property returns the current position of the cursor in the display, or can be used to change the current position. The current position is given in rows and indicates where the next text character will be displayed. To calculate the pixel offset where the cursor is located in the control window, multiply this value by the **CellHeight** property value.

See Also

Terminal Class | SocketTools Namespace

Terminal.Display Property

Gets a value that specifies the virtual display handle allocated for the control.

[Visual Basic]

```
<Description(Description:="Returns the handle to the virtual display."), _
    Browsable(Browsable:=False)>
Public ReadOnly Property Display As Integer
```

[C#]

```
[Description(Description="Returns the handle to the virtual display.")]
[Browsable(Browsable=False)]
public int Display {get;}
```

Property Value

An integer which represents a virtual display handle. If there is no current virtual display, a value of -1 is returned.

Remarks

The **Display** property returns the numeric descriptor for the virtual display created by the control. This is a read-only property value used for diagnostic purposes.

See Also

Terminal Class | SocketTools Namespace

Terminal.Emulation Property

Gets and sets the emulation mode used by the control.

[Visual Basic]

```
<Description(Description:="Set or return the emulation mode for the current
   display."), _  Category(Category:="Design")>
Public Property Emulation As TerminalEmulation
```

[C#]

```
[Description(Description="Set or return the emulation mode for the current
   display.")]
[Category(Category="Design")]
public Terminal.TerminalEmulation Emulation {get; set;}
```

Property Value

A TerminalEmulation enumeration value which specifies the emulation mode.

See Also

Terminal Class | SocketTools Namespace

Terminal.IsInitialized Property

Gets a value which indicates if the current instance of the control has been initialized successfully.

[Visual Basic]

```
<Description(Description:="Return if the control has been initialized."), _
    Browsable(Browsable:=False)>
Public ReadOnly Property IsInitialized As Boolean
```

[C#]

```
[Description(Description="Return if the control has been initialized.")]
[Browsable(Browsable=False)]
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the control has been initialized; otherwise returns **false.**

Remarks

The **IsInitialized** property is used to determine if the current instance of the control has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the control may not be able to initialize itself.

The most common reasons that a control may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to create the control window or not being able to access the system registry.

See Also

Terminal Class | SocketTools Namespace

Terminal.KeyMap Field

Gets and sets the character sequence mapped to a special key.

[Visual Basic]

```
<Description(Description:="Set or return the character sequence mapped to a special
   key."), _  Browsable(Browsable:=False)>
Public ReadOnly KeyMap As DisplayKeyMap
```

[C#]

```
[Description(Description="Set or return the character sequence mapped to a special
   key.")]
[Browsable(Browsable=False)]
public readonly DisplayKeyMap KeyMap;
```

Remarks

The **KeyMap** array allows the application to define character sequences that should be mapped to special keys. When a special key is pressed in the emulation window and there is an entry for it in the key map, the **KeyMapped** event is fired.

The array index identifies the key which will be mapped. Refer to the TerminalKey enumeration for a list of keys and their corresponding values which may be mapped by the application.

See Also

Terminal Class | SocketTools Namespace

Terminal.LastError Property

Gets or sets a value which specifies the last error that has occurred.

[Visual Basic]

```
<Description(Description:="Set or return the last error code for this instance of
   the control."), _  Browsable(Browsable:=False)>
Public Property LastError As ErrorCode
```

[C#]

```
[Description(Description="Set or return the last error code for this instance of the
   control.")]
[Browsable(Browsable=False)]
public Terminal.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the control. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

Terminal Class | SocketTools Namespace

Terminal.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
<Description(Description:="Return a description of the last error that occurred."), 
    _  Browsable(Browsable:=False)>
Public ReadOnly Property LastErrorString As String
```

[C#]

```
[Description(Description="Return a description of the last error that occurred.")]
[Browsable(Browsable=False)]
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the control. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

Terminal Class | SocketTools Namespace

Terminal.MouseX Property

Return the current mouse position in the display.

[Visual Basic]

```
<Description(Description:="Return the current mouse pointer position in the
   display."), _  Browsable(Browsable:=False)>
Public ReadOnly Property MouseX As Integer
```

[C#]

```
[Description(Description="Return the current mouse pointer position in the
   display.")]
[Browsable(Browsable=False)]
public int MouseX {get;}
```

Property Value

An integer value which specifies the column where the mouse pointer is positioned.

Remarks

The **MouseX** property returns the current position of the mouse pointer in the display. The current position is given in columns, not pixels. To calculate the pixel offset where the mouse pointer is located in the control window, multiply this value by the **CellWidth** property value.

See Also

Terminal Class | SocketTools Namespace

Terminal.MouseY Property

Return the current mouse position in the display.

[Visual Basic]

```
<Browsable(Browsable:=False), _  Description(Description:="Return the current mouse
   pointer position in the display.")>
Public ReadOnly Property MouseY As Integer
```

[C#]

```
[Browsable(Browsable=False)]
[Description(Description="Return the current mouse pointer position in the
   display.")]
public int MouseY {get;}
```

Property Value

An integer value which specifies the row where the mouse pointer is positioned.

Remarks

The **MouseY** property returns the current position of the mouse pointer in the display. The current position is given in rows, not pixels. To calculate the pixel offset where the mouse pointer is located in the control window, multiply this value by the **CellHeight** property value.

See Also

Terminal Class | SocketTools Namespace

Terminal.NewLine Property

Gets and sets the newline mode for the virtual display.

[Visual Basic]

```
<Description(Description:="Determine how carriage returns and linefeeds are
   displayed."), _  Category(Category:="Design")>
Public Property NewLine As TerminalNewLine
```

[C#]

```
[Description(Description="Determine how carriage returns and linefeeds are
   displayed.")]
[Category(Category="Design")]
public Terminal.TerminalNewLine NewLine {get; set;}
```

Property Value

A TerminalNewLine enumeration value which specifies the current newline mode.

Remarks

The **NewLine** property controls how carriage returns and linefeeds are processed by the emulator. The default is for a carriage return to position the cursor to the first column, and a linefeed to advance the cursor to the next row, scrolling the display if necessary.

See Also

Terminal Class | SocketTools Namespace

Terminal.Rows Property

Gets and sets the number of rows in the virtual display.

[Visual Basic]

```
<Description(Description:="Set or return the number of rows in the emulation
   display."), _  Category(Category:="Layout")>
Public Property Rows As Integer
```

[C#]

```
[Description(Description="Set or return the number of rows in the emulation
   display.")]
[Category(Category="Layout")]
public int Rows {get; set;}
```

Property Value

An integer value which specifies the number of rows.

Remarks

The **Rows** property returns the number of rows in the virtual display, or allows the application to change the number of rows. Currently, the number of rows may only be set to 24 or 25. Note that changing the number of rows in the display causes the current display to be invalidated, and the window will be cleared.

See Also

Terminal Class | SocketTools Namespace

Terminal.ScrollBars Property

Gets and sets the scrollbar display mode for the control.

[Visual Basic]

```
<Description(Description:="Returns or sets a value indicating whether the control
   has horizontal or vertical scroll bars."), _  Category(Category:="Design")>
Public Property ScrollBars As TerminalScrollBars
```

[C#]

```
[Description(Description="Returns or sets a value indicating whether the control has
   horizontal or vertical scroll bars.")]
[Category(Category="Design")]
public Terminal.TerminalScrollBars ScrollBars {get; set;}
```

Property Value

A TerminalScrollBars enumeration value which specifies how scrollbars will be displayed.

Remarks

The **ScrollBars** property determines what kind of scroll bars are displayed if the virtual display is larger than the emulation control's window. Scrollbars are only displayed if needed. If the emulation window is large enough to display all of the columns and rows, no scrollbars will be drawn even if they are enabled using this property.

See Also

Terminal Class | SocketTools Namespace

Terminal.SelLength Property

Gets and sets the number of characters selected in the virtual display.

[Visual Basic]

```
<Description(Description:="Set or return the number of characters selected."), _
    Browsable(Browsable:=False)>
Public Property SelLength As Integer
```

[C#]

```
[Description(Description="Set or return the number of characters selected.")]
[Browsable(Browsable=False)]
public int SelLength {get; set;}
```

Property Value

An integer value which specifies the number of selected characters.

Remarks

The **SelLength** property is used to return the number of characters currently selected in the virtual display. When used in conjunction with the **SelStart** property, it can be used to select text from the display.

See Also

Terminal Class | SocketTools Namespace

Terminal.SelStart Property

Gets and sets the starting position of the current text selection.

[Visual Basic]

```
<Description(Description:="Set or return the starting position of the current text
   selection."), _  Browsable(Browsable:=False)>
Public Property SelStart As Integer
```

[C#]

```
[Description(Description="Set or return the starting position of the current text
   selection.")]
[Browsable(Browsable=False)]
public int SelStart {get; set;}
```

Property Value

An integer value which specifies the starting position for the selected text.

Remarks

The **SelStart** property specifies an offset which is the starting position of the selected text. This property can be used in conjunction with the **SelLength** property to select text in the virtual display.

To convert the cursor position to an offset, multiply the y-position by the number of columns and add the x-position. The **SelLength** property determines the number of characters to copy from the starting position. Reading the **SelText** property returns the text displayed at the selected location.

See Also

Terminal Class | SocketTools Namespace

Terminal.SelText Property

Gets the selected text or text from a specific portion of the display.

[Visual Basic]

```
<Browsable(Browsable:=False), _  Description(Description:="Returns the selected text
   or text from a specific portion of the display.")>
Public ReadOnly Property SelText As String
```

[C#]

```
[Browsable(Browsable=False)]
[Description(Description="Returns the selected text or text from a specific portion
   of the display.")]
public string SelText {get;}
```

Property Value

A string which contains the currently selected text.

Remarks

The **SelText** property returns the text which is currently selected in the virtual display. If no text has been selected and the **SelLength** property is greater than zero, then the text starting at the position specified by the **SelStart** property will be returned.

To read a single character a specific location in the display, it is preferable to use the **Cell** property rather than calculating the offset, setting the **SelStart** property and then reading the **SelText** property.

See Also

Terminal Class | SocketTools Namespace

Terminal.Text Property

Gets and sets the text displayed by the control.

[Visual Basic]
```
<Description(Description:="Sets or returns the text displayed by the control."), _
    Category(Category:="Appearance")>
Overrides Public Property Text As String
```

[C#]
```
[Description(Description="Sets or returns the text displayed by the control.")]
[Category(Category="Appearance")]
public override string Text {get; set;}
```

Property Value

A string which contains the current display text.

Remarks

The **Text** property returns the text displayed by the control. The string that is returned contains each row of text in the display, terminated with a carriage-return linefeed. Empty cells at the end of a row are ignored so there are no extraneous spaces in the text. In other words, the value returned by the **Text** property is similar to how text is returned from a multi-line edit control. If you need to access a character at a specific location in the display, use the **Cell** property. If you need to access the text in a specific part of the display, including any empty cells, then use the **SelStart**, **SelLength** and **SelText** properties instead.

Setting the **Text** property will cause the current contents of the display to be replaced by the contents of the specified string. If the string contains escape sequences and/or control characters, they will be processed according to how the **Emulation** property is set.

See Also

Terminal Class | SocketTools Namespace

Terminal.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]

```
<Browsable(Browsable:=False), _  Description(Description:="Set or return if the
   control will throw an exception when an error occurs.")>
Public Property ThrowError As Boolean
```

[C#]

```
[Browsable(Browsable=False)]
[Description(Description="Set or return if the control will throw an exception when
   an error occurs.")]
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false.**

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

Terminal Class | SocketTools Namespace

Terminal.Version Property

Gets a value which returns the current version of the Terminal control.

[Visual Basic]

```
<Description(Description:="Return the current version of the object."), _
    Browsable(Browsable:=False)>
Public ReadOnly Property Version As String
```

[C#]

```
[Description(Description="Return the current version of the object.")]
[Browsable(Browsable=False)]
public string Version {get;}
```

Property Value

A string which specifies the version of the control.

Remarks

The **Version** property returns a string which identifies the current version and build of the Terminal control. This value can be used by an application for validation and debugging purposes.

See Also

Terminal Class | SocketTools Namespace

Terminal Methods

The methods of the **Terminal** class are listed below. For a complete list of **Terminal** class members, see the Terminal Members topic.

Public Instance Methods

BeginInvoke (inherited from Control)	Overloaded. Executes the specified delegate asynchronously with the specified arguments, on the thread that the control's underlying handle was created on.
BringToFront (inherited from Control)	Brings the control to the front of the z-order.
Clear	Clear the terminal emulation window.
ClearEol	Erase all characters from the current column to the end of the line.
Contains (inherited from Control)	Retrieves a value indicating whether the specified control is a child of the control.
CreateControl (inherited from Control)	Forces the creation of the control, including the creation of the handle and any child controls.
CreateGraphics (inherited from Control)	Creates the Graphics object for the control.
CreateObjRef (inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
DelLine	Delete the current line in the terminal emulation display.
Deselect	Deselects any selected text in the display.
Dispose (inherited from Component)	Overloaded. Releases all resources used by the Component.
DoDragDrop (inherited from Control)	Begins a drag-and-drop operation.
EndInvoke (inherited from Control)	Retrieves the return value of the asynchronous operation represented by the IAsyncResult object passed.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
FindForm (inherited from Control)	Retrieves the form that the control is on.
Focus (inherited from Control)	Sets input focus to the control.
GetChildAtPoint (inherited from Control)	Retrieves the child control that is located at the specified coordinates.
GetContainerControl (inherited from Control)	Returns the next ContainerControl up the control's chain of parent controls.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.

GetLifetimeService (inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetNextControl (inherited from Control)	Retrieves the next control forward or back in the tab order of child controls.
GetType (inherited from Object)	Gets the Type of the current instance.
Hide (inherited from Control)	Conceals the control from the user.
Initialize	Overloaded. Initialize an instance of the Terminal control.
InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
InsLine	Insert an empty line at the current position in the terminal emulation display.
Invalidate (inherited from Control)	Overloaded. Invalidates the specified region of the control (adds it to the control's update region, which is the area that will be repainted at the next paint operation), and causes a paint message to be sent to the control.
Invoke (inherited from Control)	Overloaded. Executes the specified delegate, on the thread that owns the control's underlying window handle, with the specified list of arguments.
PerformLayout (inherited from Control)	Overloaded. Forces the control to apply layout logic to all its child controls.
PointToClient (inherited from Control)	Computes the location of the specified screen point into client coordinates.
PointToScreen (inherited from Control)	Computes the location of the specified client point into screen coordinates.
PreProcessMessage	Preprocesses input messages within the message loop before they are dispatched.
RectangleToClient (inherited from Control)	Computes the size and location of the specified screen rectangle in client coordinates.
RectangleToScreen (inherited from Control)	Computes the size and location of the specified client rectangle in screen coordinates.
Refresh	Forces a complete redraw of the virtual display.
Reset	Reset the internal state of the control, resetting all properties to their default values.
ResetBackColor (inherited from Control)	Resets the BackColor property to its default value.
ResetBindings (inherited from Control)	Resets the DataBindings property to its default value.
ResetCursor (inherited from Control)	Resets the Cursor property to its default value.
ResetFont (inherited from Control)	Resets the Fontproperty to its default value.
ResetForeColor (inherited from Control)	Resets the ForeColor property to its default value.

ResetImeMode (inherited from Control)	Resets the ImeMode property to its default value.
ResetRightToLeft (inherited from Control)	Resets the RightToLeft property to its default value.
ResetText (inherited from Control)	Resets the Text property to its default value.
ResumeLayout (inherited from Control)	Overloaded. Resumes normal layout logic.
Scale (inherited from Control)	Overloaded. Scales the control and any child controls to the specified ratio.
ScrollControlIntoView (inherited from ScrollableControl)	
ScrollDown	Scroll the display down by one line.
ScrollUp	Scroll the display up by one line.
Select (inherited from Control)	Overloaded. Activates the control.
SelectNextControl (inherited from Control)	Activates the next control.
SelectText	Overloaded. Selects a region of the virtual display and returns the selected text.
SendToBack (inherited from Control)	Sends the control to the back of the z-order.
SetAutoScrollMargin (inherited from ScrollableControl)	Sets the size of the auto-scroll margins.
SetBounds (inherited from Control)	Overloaded. Sets the bounds of the control to the specified location and size.
Show (inherited from Control)	Displays the control to the user.
SuspendLayout (inherited from Control)	Temporarily suspends the layout logic for the control.
ToString (inherited from Component)	
Uninitialize	Uninitialize the control and release any resources allocated for the current process.
Update (inherited from Control)	Causes the control to redraw the invalidated regions within its client area.
Validate (inherited from ContainerControl)	Validates the last invalidated control and its ancestors up through, but not including, the current control.
Write	Overloaded. Write the contents of a byte array to the virtual display.

Protected Instance Methods

AccessibilityNotifyClients (inherited from Control)	Notifies the accessibility client applications of the specified AccessibleEvents for the specified child control.
AdjustFormScrollbars (inherited from ContainerControl)	
CreateAccessibilityInstance (inherited from	Creates a new accessibility object for the control.

Control)	
CreateControlsInstance (inherited from Control)	Creates a new instance of the control collection for the control.
CreateHandle (inherited from Control)	Creates a handle for the control.
DefWndProc (inherited from Control)	Sends the specified message to the default window procedure.
DestroyHandle (inherited from Control)	Destroys the handle associated with the control.
Dispose	Overloaded. Releases the unmanaged resources allocated by the Terminal class and optionally releases the managed resources.
Finalize	Destroys an instance of the control, releasing the resources allocated for the virtual display.
GetScrollState (inherited from ScrollableControl)	
GetService (inherited from Component)	Returns an object that represents a service provided by the Component or by its Container.
GetStyle (inherited from Control)	Retrieves the value of the specified control style bit for the control.
GetTopLevel (inherited from Control)	Determines if the control is a top-level control.
InitLayout (inherited from Control)	Called after the control has been added to another container.
InvokeGotFocus (inherited from Control)	Raises the GotFocus event for the specified control.
InvokeLostFocus (inherited from Control)	Raises the LostFocus event for the specified control.
InvokeOnClick (inherited from Control)	Raises the Click event for the specified control.
InvokePaint (inherited from Control)	Raises the Paint event for the specified control.
InvokePaintBackground (inherited from Control)	Raises the PaintBackground event for the specified control.
IsInputChar (inherited from Control)	Determines if a character is an input character that the control recognizes.
IsInputKey (inherited from Control)	Determines whether the specified key is a regular input key or a special key that requires preprocessing.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.
NotifyInvalidate (inherited from Control)	
OnBackColorChanged	Raises the BackColorChanged event.
OnBackgroundImageChanged (inherited from Control)	Raises the BackgroundImageChanged event.
OnBindingContextChanged (inherited from Control)	Raises the BindingContextChangedevent.
OnCausesValidationChanged (inherited from	Raises the CausesValidationChangedevent.

Control)	
OnChangeUICues (inherited from Control)	Raises the ChangeUICues event.
OnClick (inherited from Control)	Raises the Click event.
OnContextMenuChanged (inherited from Control)	Raises the ContextMenuChangedevent.
OnControlAdded (inherited from Control)	Raises the ControlAdded event.
OnControlRemoved (inherited from ContainerControl)	
OnCreateControl (inherited from UserControl)	
OnCursorChanged (inherited from Control)	Raises the CursorChangedevent.
OnDockChanged (inherited from Control)	Raises the DockChangedevent.
OnDoubleClick (inherited from Control)	Raises the DoubleClick event.
OnDragDrop (inherited from Control)	Raises the DragDrop event.
OnDragEnter (inherited from Control)	Raises the DragEnter event.
OnDragLeave (inherited from Control)	Raises the DragLeave event.
OnDragOver (inherited from Control)	Raises the DragOver event.
OnEnabledChanged (inherited from Control)	Raises the EnabledChanged event.
OnEnter (inherited from Control)	Raises the Enter event.
OnFontChanged	Raises the FontChanged event.
OnForeColorChanged	Raises the ForeColorChanged event.
OnGiveFeedback (inherited from Control)	Raises the GiveFeedback event.
OnGotFocus	Raises the GotFocus event.
OnHandleCreated (inherited from Control)	Raises the HandleCreated event.
OnHandleDestroyed (inherited from Control)	Raises the HandleDestroyed event.
OnHelpRequested (inherited from Control)	Raises the HelpRequested event.
OnImeModeChanged (inherited from Control)	Raises the ImeModeChanged event.
OnInvalidated (inherited from Control)	Raises the Invalidated event.
OnKeyDown (inherited from Control)	Raises the KeyDown event.
OnKeyPress (inherited from Control)	Raises the KeyPress event.
OnKeyUp (inherited from Control)	Raises the KeyUp event.
OnLayout (inherited from ScrollableControl)	
OnLeave (inherited from Control)	Raises the Leave event.
OnLoad (inherited from UserControl)	Raises the Load event.
OnLocationChanged (inherited from Control)	Raises the LocationChanged event.
OnLostFocus	Raises the LostFocus event.
OnMouseDown	Raises the MouseDown event.

OnMouseEnter (inherited from Control)	Raises the MouseEnter event.
OnMouseHover (inherited from Control)	Raises the MouseHover event.
OnMouseLeave (inherited from Control)	Raises the MouseLeave event.
OnMouseMove	Raises the MouseMove event.
OnMouseUp	Raises the MouseUp event.
OnMouseWheel (inherited from ScrollableControl)	
OnMove (inherited from Control)	Raises the Move event.
OnNotifyMessage (inherited from Control)	Notifies the control of Windows messages.
OnPaint	Raises the Paint event.
OnPaintBackground (inherited from Control)	Paints the background of the control.
OnParentBackColorChanged (inherited from Control)	Raises the BackColorChanged event when the BackColor property value of the control's container changes.
OnParentBackgroundImageChanged (inherited from Control)	Raises the BackgroundImageChanged event when the BackgroundImage property value of the control's container changes.
OnParentBindingContextChanged (inherited from Control)	Raises the BindingContextChanged event when the BindingContext property value of the control's container changes.
OnParentChanged	Raises the ParentChanged event.
OnParentEnabledChanged (inherited from Control)	Raises the EnabledChanged event when the Enabled property value of the control's container changes.
OnParentFontChanged (inherited from Control)	Raises the FontChanged event when the Font property value of the control's container changes.
OnParentForeColorChanged (inherited from Control)	Raises the ForeColorChanged event when the ForeColor property value of the control's container changes.
OnParentRightToLeftChanged (inherited from Control)	Raises the RightToLeftChanged event when the RightToLeft property value of the control's container changes.
OnParentVisibleChanged (inherited from Control)	Raises the VisibleChanged event when the Visible property value of the control's container changes.
OnQueryContinueDrag (inherited from Control)	Raises the QueryContinueDrag event.
OnResize	Raises the Resize event.
OnRightToLeftChanged (inherited from Control)	Raises the RightToLeftChangedevent.
OnSizeChanged (inherited from Control)	Raises the SizeChanged event.
OnStyleChanged (inherited from Control)	Raises the StyleChanged event.
OnSystemColorsChanged (inherited from Control)	Raises the SystemColorsChanged event.

OnTabIndexChanged (inherited from Control)	Raises the TabIndexChanged event.
OnTabStopChanged (inherited from Control)	Raises the TabStopChanged event.
OnTextChanged (inherited from Control)	Raises the TextChanged event.
OnValidated (inherited from Control)	Raises the Validated event.
OnValidating (inherited from Control)	Raises the Validating event.
OnVisibleChanged (inherited from ScrollableControl)	
ProcessCmdKey (inherited from Control)	Processes a command key.
ProcessDialogChar (inherited from ContainerControl)	
ProcessDialogKey (inherited from ContainerControl)	
ProcessKeyEventArgs (inherited from Control)	Processes a key message and generates the appropriate control events.
ProcessKeyPreview (inherited from Control)	Previews a keyboard message.
ProcessMnemonic (inherited from ContainerControl)	
ProcessTabKey (inherited from ContainerControl)	Selects the next available control and makes it the active control.
RaiseDragEvent (inherited from Control)	
RaiseKeyEvent (inherited from Control)	
RaiseMouseEvent (inherited from Control)	
RaisePaintEvent (inherited from Control)	
RecreateHandle (inherited from Control)	Forces the re-creation of the handle for the control.
ResetMouseEventArgs (inherited from Control)	
RtlTranslateAlignment (inherited from Control)	Overloaded. Converts the specified HorizontalAlignment to the appropriate HorizontalAlignment to support right-to-left text.
RtlTranslateContent (inherited from Control)	Converts the specified ContentAlignment to the appropriate ContentAlignment to support right-to-left text.
RtlTranslateHorizontal (inherited from Control)	Converts the specified HorizontalAlignment to the appropriate HorizontalAlignment to support right-to-left text.
RtlTranslateLeftRight (inherited from Control)	Converts the specified LeftRightAlignment to the appropriate LeftRightAlignment to support right-to-left text.
ScaleCore (inherited from ScrollableControl)	
Select (inherited from ContainerControl)	Overloaded.

SetBoundsCore (inherited from Control)	Performs the work of setting the specified bounds of this control.
SetClientSizeCore (inherited from Control)	Sets the size of the client area of the control.
SetDisplayRectLocation (inherited from ScrollableControl)	
SetScrollState (inherited from ScrollableControl)	
SetStyle (inherited from Control)	Sets the specified style bit to the specified value.
SetTopLevel (inherited from Control)	Sets the control as the top-level control.
SetVisibleCore (inherited from Control)	Sets the control to the specified visible state.
UpdateBounds (inherited from Control)	Overloaded. Updates the bounds of the control with the current size and location.
UpdateDefaultButton (inherited from ContainerControl)	
UpdateStyles (inherited from Control)	Forces the assigned styles to be reapplied to the control.
UpdateZOrder (inherited from Control)	Updates the control in its parent's z-order.
WndProc	Processes Windows messages.

Protected Internal Instance Methods

ProcessKeyMessage (inherited from Control)	Processes a keyboard message.

See Also

Terminal Class | SocketTools Namespace

Terminal.Clear Method

Clear the terminal emulation window.

[Visual Basic]

```
Public Sub Clear()
```

[C#]

```
public void Clear();
```

See Also

Terminal Class | SocketTools Namespace

Terminal.ClearEol Method

Erase all characters from the current column to the end of the line.

[Visual Basic]

```
Public Sub ClearEol()
```

[C#]

```
public void ClearEol();
```

See Also

Terminal Class | SocketTools Namespace

Terminal.DelLine Method

Delete the current line in the terminal emulation display.

[Visual Basic]

```
Public Sub DelLine()
```

[C#]

```
public void DelLine();
```

See Also

Terminal Class | SocketTools Namespace

Terminal.Deselect Method

Deselects any selected text in the display.

[Visual Basic]

```
Public Sub Deselect()
```

[C#]

```
public void Deselect();
```

Remarks

The **Deselect** method deselects any text that has been previously selected using the **Select** method, or by the user using the mouse if the **AutoSelect** property has been set to **true**.

See Also

Terminal Class | SocketTools Namespace

Terminal.Dispose Method

Releases the unmanaged resources allocated by the Terminal class and optionally releases the managed resources.

Overload List

Inherited from Component.

public void Dispose();

Releases the unmanaged resources allocated by the Terminal class and optionally releases the managed resources.

protected override void Dispose(bool);

See Also

Terminal Class | SocketTools Namespace

Terminal.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the Terminal class and optionally releases the managed resources.

[Visual Basic]

```
Overrides Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]

```
protected override void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **Terminal** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

Terminal Class | SocketTools Namespace | Terminal.Dispose Overload List

Terminal.Finalize Method

Destroys an instance of the control, releasing the resources allocated for the virtual display.

[Visual Basic]

```
Overrides Protected Sub Finalize()
```

[C#]

```
protected override void Finalize();
```

See Also

Terminal Class | SocketTools Namespace

Terminal.Initialize Method

Initialize an instance of the Terminal control.

Overload List

Initialize an instance of the Terminal control.

public bool Initialize();

Initialize an instance of the Terminal control.

public bool Initialize(string);

See Also

Terminal Class | SocketTools Namespace | Uninitialize Method

Terminal.Initialize Method ()

Initialize an instance of the Terminal control.

[Visual Basic]
```
Overloads Public Function Initialize() As Boolean
```

[C#]
```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The **Initialize** method can be used to explicitly initialize an instance of the Terminal control and allocating resources for the current thread. Typically it is not necessary to explicitly call this method because the instance of the class is initialized by the class constructor. However, if the **Uninitialize** method is called, the control must be re-initialized before any other methods are called.

See Also

Terminal Class | SocketTools Namespace | Terminal.Initialize Overload List | Uninitialize Method

Terminal.Initialize Method (String)

Initialize an instance of the Terminal control.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the control.

Return Value

A boolean value which specifies if the control was initialized successfully.

Remarks

The **Initialize** method can be used to explicitly initialize an instance of the Terminal control. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the Terminal control.

See Also

Terminal Class | SocketTools Namespace | Terminal.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

Terminal.InsLine Method

Insert an empty line at the current position in the terminal emulation display.

[Visual Basic]

```
Public Sub InsLine()
```

[C#]

```
public void InsLine();
```

See Also

Terminal Class | SocketTools Namespace

Terminal.OnBackColorChanged Method

Raises the BackColorChanged event.

[Visual Basic]
```
<EditorBrowsable(State:=EditorBrowsableState.Advanced)>
Overrides Protected Sub OnBackColorChanged( _
   ByVal e As EventArgs _
)
```

[C#]
```
[EditorBrowsable(State=EditorBrowsableState.Advanced)]
protected override void OnBackColorChanged(
   EventArgs e
);
```

Parameters

e

An EventArgs that contains the event data.

See Also

Terminal Class | SocketTools Namespace

Terminal.OnFontChanged Method

Raises the FontChanged event.

```
[Visual Basic]
<EditorBrowsable(State:=EditorBrowsableState.Advanced)>
Overrides Protected Sub OnFontChanged( _
   ByVal e As EventArgs _
)
```

```
[C#]
[EditorBrowsable(State=EditorBrowsableState.Advanced)]
protected override void OnFontChanged(
   EventArgs e
);
```

Parameters

e

An EventArgs that contains the event data.

See Also

Terminal Class | SocketTools Namespace

Terminal.OnForeColorChanged Method

Raises the ForeColorChanged event.

[Visual Basic]
```
<EditorBrowsable(State:=EditorBrowsableState.Advanced)>
Overrides Protected Sub OnForeColorChanged( _
   ByVal e As EventArgs _
)
```

[C#]
```
[EditorBrowsable(State=EditorBrowsableState.Advanced)]
protected override void OnForeColorChanged(
   EventArgs e
);
```

Parameters

e

An EventArgs that contains the event data.

See Also

Terminal Class | SocketTools Namespace

Terminal.OnGotFocus Method

Raises the GotFocus event.

[Visual Basic]

```
<EditorBrowsable(State:=EditorBrowsableState.Advanced)>
Overrides Protected Sub OnGotFocus( _
   ByVal e As EventArgs _
)
```

[C#]

```
[EditorBrowsable(State=EditorBrowsableState.Advanced)]
protected override void OnGotFocus(
   EventArgs e
);
```

Parameters

e

An EventArgs that contains the event data.

See Also

Terminal Class | SocketTools Namespace

Terminal.OnLostFocus Method

Raises the LostFocus event.

[Visual Basic]
```
<EditorBrowsable(State:=EditorBrowsableState.Advanced)>
Overrides Protected Sub OnLostFocus( _
   ByVal e As EventArgs _
)
```

[C#]
```
[EditorBrowsable(State=EditorBrowsableState.Advanced)]
protected override void OnLostFocus(
   EventArgs e
);
```

Parameters

e

An EventArgs that contains the event data.

See Also

Terminal Class | SocketTools Namespace

Terminal.OnMouseDown Method

Raises the MouseDown event.

[Visual Basic]

```
<EditorBrowsable(State:=EditorBrowsableState.Advanced)>
Overrides Protected Sub OnMouseDown( _
   ByVal e As MouseEventArgs _
)
```

[C#]

```
[EditorBrowsable(State=EditorBrowsableState.Advanced)]
protected override void OnMouseDown(
   MouseEventArgs e
);
```

Parameters

e

A MouseEventArgs that contains the event data.

See Also

Terminal Class | SocketTools Namespace

Terminal.OnMouseMove Method

Raises the MouseMove event.

[Visual Basic]

```
<EditorBrowsable(State:=EditorBrowsableState.Advanced)>
Overrides Protected Sub OnMouseMove( _
   ByVal e As MouseEventArgs _
)
```

[C#]

```
[EditorBrowsable(State=EditorBrowsableState.Advanced)]
protected override void OnMouseMove(
   MouseEventArgs e
);
```

Parameters

e

A MouseEventArgs that contains the event data.

See Also

Terminal Class | SocketTools Namespace

Terminal.OnMouseUp Method

Raises the MouseUp event.

```
[Visual Basic]
<EditorBrowsable(State:=EditorBrowsableState.Advanced)>
Overrides Protected Sub OnMouseUp( _
   ByVal e As MouseEventArgs _
)
```

```
[C#]
[EditorBrowsable(State=EditorBrowsableState.Advanced)]
protected override void OnMouseUp(
   MouseEventArgs e
);
```

Parameters

e

A MouseEventArgs that contains the event data.

See Also

Terminal Class | SocketTools Namespace

Terminal.OnPaint Method

Raises the Paint event.

[Visual Basic]

```
<EditorBrowsable(State:=EditorBrowsableState.Advanced)>
Overrides Protected Sub OnPaint( _
   ByVal e As PaintEventArgs _
)
```

[C#]

```
[EditorBrowsable(State=EditorBrowsableState.Advanced)]
protected override void OnPaint(
   PaintEventArgs e
);
```

Parameters

e

A PaintEventArgs that contains the event data.

See Also

Terminal Class | SocketTools Namespace

Terminal.OnParentChanged Method

Raises the ParentChanged event.

[Visual Basic]

```
<EditorBrowsable(State:=EditorBrowsableState.Advanced)>
Overrides Protected Sub OnParentChanged( _
   ByVal e As EventArgs _
)
```

[C#]

```
[EditorBrowsable(State=EditorBrowsableState.Advanced)]
protected override void OnParentChanged(
   EventArgs e
);
```

Parameters

e

An EventArgs that contains the event data.

See Also

Terminal Class | SocketTools Namespace

Terminal.OnResize Method

Raises the Resize event.

[Visual Basic]
```
<EditorBrowsable(State:=EditorBrowsableState.Advanced)>
Overrides Protected Sub OnResize( _
   ByVal e As EventArgs _
)
```

[C#]
```
[EditorBrowsable(State=EditorBrowsableState.Advanced)]
protected override void OnResize(
   EventArgs e
);
```

Parameters

e

An EventArgs that contains the event data.

See Also

Terminal Class | SocketTools Namespace

Terminal.PreProcessMessage Method

Preprocesses input messages within the message loop before they are dispatched.

[Visual Basic]

```
Overrides Public Function PreProcessMessage( _
   ByRef msg As Message _
) As Boolean
```

[C#]

```
public override bool PreProcessMessage(
   ref Message msg
);
```

Parameters

msg

A Message passed by reference, that represents the message to process.

Return Value

A value of **true** if the message was processed by the control; otherwise, **false**.

Remarks

PreProcessMessage is called by the application's message loop to preprocess input messages before they are dispatched. Possible values for the msg.message field are WM_KEYDOWN, WM_SYSKEYDOWN, WM_CHAR, and WM_SYSCHAR.

When overriding **preProcessMessage()**, a control should return **true** to indicate that it has processed the message. For messages that are not processed by the control, the result of **base.preProcessMessage()** should be returned. Controls will typically override one of the more specialized methods such as **isInputChar()**, **isInputKey()**, **processCmdKey()**, **processDialogChar()**, or **processDialogKey()** instead of overriding **preProcessMessage()**.

See Also

Terminal Class | SocketTools Namespace

Terminal.Refresh Method

Forces a complete redraw of the virtual display.

[Visual Basic]

```
Overrides Public Sub Refresh()
```

[C#]

```
public override void Refresh();
```

Remarks

The **Refresh** method forces the control to redraw the virtual display. Normally, the virtual display is automatically redrawn after the display has been modified and there are no other events being processed. However, there may be situations where you want the display updated immediately.

To prevent the control from automatically redrawing the window when the virtual display has been modified, set the **AutoRefresh** property to **false**. You can then call the **Refresh** method to force the control to be redrawn as needed

See Also

Terminal Class | SocketTools Namespace

Terminal.Reset Method

Reset the internal state of the control, resetting all properties to their default values.

[Visual Basic]
```
Public Sub Reset()
```

[C#]
```
public void Reset();
```

See Also

Terminal Class | SocketTools Namespace

Terminal.ScrollDown Method

Scroll the display down by one line.

[Visual Basic]

```
Public Sub ScrollDown()
```

[C#]

```
public void ScrollDown();
```

See Also

Terminal Class | SocketTools Namespace

Terminal.ScrollUp Method

Scroll the display up by one line.

[Visual Basic]

```
Public Sub ScrollUp()
```

[C#]

```
public void ScrollUp();
```

See Also

Terminal Class | SocketTools Namespace

Terminal.SelectText Method

Selects a region of the virtual display and returns the selected text.

Overload List

Selects a region of the virtual display and returns the selected text.

public string SelectText();

Selects a region of the virtual display and returns the selected text.

public string SelectText(int,int);

Selects a region of the virtual display and returns the selected text.

public string SelectText(int,int,int,int);

Selects a region of the virtual display and returns the selected text.

public string SelectText(int,int,int,int,TerminalSelection);

See Also

Terminal Class | SocketTools Namespace

Terminal.SelectText Method ()

Selects a region of the virtual display and returns the selected text.

[Visual Basic]
```
Overloads Public Function SelectText() As String
```

[C#]
```
public string SelectText();
```

Return Value

A string which contains the selected text.

Remarks

The **Select** method selects the entire contents of the virtual display. If the method fails because incorrect row or column values were used, or because an invalid option was specified, it will return an empty string.

See Also

Terminal Class | SocketTools Namespace | Terminal.SelectText Overload List

Terminal.SelectText Method (Int32, Int32)

Selects a region of the virtual display and returns the selected text.

[Visual Basic]

```
Overloads Public Function SelectText( _
   ByVal column As Integer, _
   ByVal row As Integer _
) As String
```

[C#]

```
public string SelectText(
   int column,
   int row
);
```

Parameters

column

An integer value which specifies the starting column for the text selection.

row

An integer value which specifies the starting row for the text selection.

Return Value

A string which contains the selected text.

Remarks

The **Select** method selects a region of the virtual display, from the specified column and row to the end of the virtual display. This enables the application to select text in the same way that a user would by clicking and dragging the mouse over the display window. If the method fails because incorrect row or column values were used, or because an invalid option was specified, it will return an empty string.

See Also

Terminal Class | SocketTools Namespace | Terminal.SelectText Overload List

Terminal.SelectText Method (Int32, Int32, Int32, Int32)

Selects a region of the virtual display and returns the selected text.

[Visual Basic]

```
Overloads Public Function SelectText( _
   ByVal column1 As Integer, _
   ByVal row1 As Integer, _
   ByVal column2 As Integer, _
   ByVal row2 As Integer _
) As String
```

[C#]

```
public string SelectText(
   int column1,
   int row1,
   int column2,
   int row2
);
```

Parameters

column1
An integer value which specifies the starting column for the text selection.

row1
An integer value which specifies the starting row for the text selection.

column2
An integer value which specifies the ending column for the text selection.

row2
An integer value which specifies the ending row for the text selection.

Return Value

A string which contains the selected text.

Remarks

The **Select** method selects a region of the virtual display. This enables the application to select text in the same way that a user would by clicking and dragging the mouse over the display window. If the method fails because incorrect row or column values were used, or because an invalid option was specified, it will return an empty string.

See Also

Terminal Class | SocketTools Namespace | Terminal.SelectText Overload List

Terminal.SelectText Method (Int32, Int32, Int32, Int32, TerminalSelection)

Selects a region of the virtual display and returns the selected text.

[Visual Basic]

```
Overloads Public Function SelectText( _
   ByVal column1 As Integer, _
   ByVal row1 As Integer, _
   ByVal column2 As Integer, _
   ByVal row2 As Integer, _
   ByVal options As TerminalSelection _
) As String
```

[C#]

```
public string SelectText(
   int column1,
   int row1,
   int column2,
   int row2,
   TerminalSelection options
);
```

Parameters

column1

An integer value which specifies the starting column for the text selection.

row1

An integer value which specifies the starting row for the text selection.

column2

An integer value which specifies the ending column for the text selection.

row2

An integer value which specifies the ending row for the text selection.

options

A TerminalSelection enumeration value which specifies one or more selection options.

Return Value

A string which contains the selected text.

Remarks

The **Select** method selects a region of the virtual display. This enables the application to select text in the same way that a user would by clicking and dragging the mouse over the display window. If the method fails because incorrect row or column values were used, or because an invalid option was specified, it will return an empty string.

See Also

Terminal Class | SocketTools Namespace | Terminal.SelectText Overload List

Terminal.Uninitialize Method

Uninitialize the control and release any resources allocated for the current process.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** destroys the virtual display and releases resources allocated for the current thread. After this method has been called, the control must re-initialized before it can be used.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the control is no longer needed.

See Also

Terminal Class | SocketTools Namespace

Terminal.WndProc Method

Processes Windows messages.

```
[Visual Basic]
<EditorBrowsable(State:=EditorBrowsableState.Advanced)>
Overrides Protected Sub WndProc( _
   ByRef m As Message _
)
```

```
[C#]
[EditorBrowsable(State=EditorBrowsableState.Advanced)]
protected override void WndProc(
   ref Message m
);
```

Parameters

m

The Windows message to process.

Remarks

All messages are sent to the **WndProc** method after getting filtered through the PreProcessMessage method. The **WndProc** method corresponds exactly to the Windows WindowProc function. For more information about processing Windows messages, see the WindowProc function documentation in the Windows Platform SDK reference located in the MSDN Library.

See Also

Terminal Class | SocketTools Namespace

Terminal.Write Method

Write the contents of a byte array to the virtual display.

Overload List

Write the contents of a byte array to the virtual display.

public bool Write(byte[]);

Write the contents of a byte array to the virtual display.

public bool Write(byte[],int);

Write a character to the virtual display.

public bool Write(char);

Write one or more characters to the virtual display.

public bool Write(char,int);

Write the contents of a string to the virtual display.

public bool Write(string);

Write the contents of a string to the virtual display.

public bool Write(string,int);

See Also

Terminal Class | SocketTools Namespace

Terminal.Write Method (Byte[])

Write the contents of a byte array to the virtual display.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As Byte() _
) As Boolean
```

[C#]

```
public bool Write(
   byte[] buffer
);
```

Parameters

buffer
A byte array which contains the data to be written to the virtual display.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred.

Remarks

The **Write** method writes the data in the specified byte array to the virtual display at the current cursor location. If the data contains control characters or escape sequences, they will be processed according the **Emulation** property setting.

See Also

Terminal Class | SocketTools Namespace | Terminal.Write Overload List

Terminal.Write Method (Byte[], Int32)

Write the contents of a byte array to the virtual display.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool Write(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array which contains the data to be written to the virtual display.

length

An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred.

Remarks

The **Write** method writes the data in the specified byte array to the virtual display at the current cursor location. If the data contains control characters or escape sequences, they will be processed according the **Emulation** property setting.

See Also

Terminal Class | SocketTools Namespace | Terminal.Write Overload List

Terminal.Write Method (Char)

Write a character to the virtual display.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal character As Char _
) As Boolean
```

[C#]

```
public bool Write(
   char character
);
```

Parameters

character

A character which will be written to the virtual display.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred.

Remarks

The **Write** method writes one character to the virtual display at the current cursor location. If a control characters is specified, it will be processed according the **Emulation** property setting.

See Also

Terminal Class | SocketTools Namespace | Terminal.Write Overload List

Terminal.Write Method (Char, Int32)

Write one or more characters to the virtual display.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal character As Char, _
   ByVal repeat As Integer _
) As Boolean
```

[C#]

```
public bool Write(
   char character,
   int repeat
);
```

Parameters

character

A character which will be written to the virtual display.

repeat

The number of characters that will be written.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred.

Remarks

The **Write** method writes the specified character to the virtual display at the current cursor location. If one or more control characters are specified, they will be processed according the **Emulation** property setting.

See Also

Terminal Class | SocketTools Namespace | Terminal.Write Overload List

Terminal.Write Method (String)

Write the contents of a string to the virtual display.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String _
) As Boolean
```

[C#]

```
public bool Write(
   string buffer
);
```

Parameters

buffer
A string which contains the data to be written to the virtual display.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred.

Remarks

The **Write** method writes the data in the specified string to the virtual display at the current cursor location. If the data contains control characters or escape sequences, they will be processed according the **Emulation** property setting.

See Also

Terminal Class | SocketTools Namespace | Terminal.Write Overload List

Terminal.Write Method (String, Int32)

Write the contents of a string to the virtual display.

[Visual Basic]

```
Overloads Public Function Write( _
   ByVal buffer As String, _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool Write(
   string buffer,
   int length
);
```

Parameters

buffer

A string which contains the data to be written to the virtual display.

length

An integer value which specifies the maximum number of bytes of data to write. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

A boolean value which specifies if the operation completed successfully. A return value of **false** indicates an error has occurred.

Remarks

The **Write** method writes the data in the specified string to the virtual display at the current cursor location. If the data contains control characters or escape sequences, they will be processed according the **Emulation** property setting.

See Also

Terminal Class | SocketTools Namespace | Terminal.Write Overload List

Terminal Events

The events of the **Terminal** class are listed below. For a complete list of **Terminal** class members, see the Terminal Members topic.

Public Instance Events

BackColorChanged (inherited from Control)	Occurs when the value of the BackColor property changes.
BackgroundImageChanged (inherited from Control)	Occurs when the value of the BackgroundImage property changes.
BindingContextChanged (inherited from Control)	Occurs when the value of the BindingContext property changes.
CausesValidationChanged (inherited from Control)	Occurs when the value of the CausesValidation property changes.
ChangeUICues (inherited from Control)	Occurs when the focus or keyboard user interface (UI) cues change.
Click (inherited from Control)	Occurs when the control is clicked.
ContextMenuChanged (inherited from Control)	Occurs when the value of the ContextMenu property changes.
ControlAdded (inherited from Control)	Occurs when a new control is added to the ControlCollection .
ControlRemoved (inherited from Control)	Occurs when a control is removed from the ControlCollection .
CursorChanged (inherited from Control)	Occurs when the value of the Cursor property changes.
Disposed (inherited from Component)	Adds an event handler to listen to the Disposed event on the component.
DockChanged (inherited from Control)	Occurs when the value of the Dock property changes.
DoubleClick (inherited from Control)	Occurs when the control is double-clicked.
DragDrop (inherited from Control)	Occurs when a drag-and-drop operation is completed.
DragEnter (inherited from Control)	Occurs when an object is dragged into the control's bounds.
DragLeave (inherited from Control)	Occurs when an object is dragged out of the control's bounds.
DragOver (inherited from Control)	Occurs when an object is dragged over the control's bounds.
EnabledChanged (inherited from Control)	Occurs when the Enabled property value has changed.
Enter (inherited from Control)	Occurs when the control is entered.
FontChanged (inherited from Control)	Occurs when the Font property value changes.

⚡ ForeColorChanged (inherited from Control)	Occurs when the ForeColor property value changes.
⚡ GiveFeedback (inherited from Control)	Occurs during a drag operation.
⚡ GotFocus (inherited from Control)	Occurs when the control receives focus.
⚡ HandleCreated (inherited from Control)	Occurs when a handle is created for the control.
⚡ HandleDestroyed (inherited from Control)	Occurs when the control's handle is in the process of being destroyed.
⚡ HelpRequested (inherited from Control)	Occurs when the user requests help for a control.
⚡ ImeModeChanged (inherited from Control)	Occurs when the ImeMode property has changed.
⚡ Invalidated (inherited from Control)	Occurs when a control's display requires redrawing.
⚡ KeyDown (inherited from Control)	Occurs when a key is pressed while the control has focus.
⚡ KeyMapped	Occurs when a mapped key is pressed.
⚡ KeyPress (inherited from Control)	Occurs when a key is pressed while the control has focus.
⚡ KeyUp (inherited from Control)	Occurs when a key is released while the control has focus.
⚡ Layout (inherited from Control)	Occurs when a control should reposition its child controls.
⚡ Leave (inherited from Control)	Occurs when the input focus leaves the control.
⚡ Load (inherited from UserControl)	Occurs before the control becomes visible for the first time.
⚡ LocationChanged (inherited from Control)	Occurs when the Location property value has changed.
⚡ LostFocus (inherited from Control)	Occurs when the control loses focus.
⚡ MouseDown (inherited from Control)	Occurs when the mouse pointer is over the control and a mouse button is pressed.
⚡ MouseEnter (inherited from Control)	Occurs when the mouse pointer enters the control.
⚡ MouseHover (inherited from Control)	Occurs when the mouse pointer hovers over the control.
⚡ MouseLeave (inherited from Control)	Occurs when the mouse pointer leaves the control.
⚡ MouseMove (inherited from Control)	Occurs when the mouse pointer is moved over the control.
⚡ MouseUp (inherited from Control)	Occurs when the mouse pointer is over the control and a mouse button is released.
⚡ MouseWheel (inherited from Control)	Occurs when the mouse wheel moves while the control has focus.
⚡ Move (inherited from Control)	Occurs when the control is moved.
⚡ Paint (inherited from Control)	Occurs when the control is redrawn.

⚡ ParentChanged (inherited from Control)	Occurs when the Parent property value changes.
⚡ QueryAccessibilityHelp (inherited from Control)	Occurs when AccessibleObject is providing help to accessibility applications.
⚡ QueryContinueDrag (inherited from Control)	Occurs during a drag-and-drop operation and allows the drag source to determine whether the drag-and-drop operation should be canceled.
⚡ Resize (inherited from Control)	Occurs when the control is resized.
⚡ RightToLeftChanged (inherited from Control)	Occurs when the RightToLeft property value changes.
⚡ SizeChanged (inherited from Control)	Occurs when the Size property value changes.
⚡ StyleChanged (inherited from Control)	Occurs when the control style changes.
⚡ SystemColorsChanged (inherited from Control)	Occurs when the system colors change.
⚡ TabIndexChanged (inherited from Control)	Occurs when the TabIndex property value changes.
⚡ TabStopChanged (inherited from Control)	Occurs when the TabStop property value changes.
⚡ TextChanged (inherited from UserControl)	
⚡ Validated (inherited from Control)	Occurs when the control is finished validating.
⚡ Validating (inherited from Control)	Occurs when the control is validating.
⚡ VisibleChanged (inherited from Control)	Occurs when the Visible property value changes.

See Also

Terminal Class | SocketTools Namespace

Terminal.KeyMapped Event

Occurs when a mapped key is pressed.

[Visual Basic]
```
Public Event KeyMapped As KeyMappedEventHandler
```

[C#]
```
public event KeyMappedEventHandler KeyMapped;
```

Event Data

The event handler receives an argument of type Terminal.KeyMappedEventArgs containing data related to this event. The following **Terminal.KeyMappedEventArgs** properties provide information specific to this event.

Property	Description
Alt	Gets a value indicating whether the ALT key was pressed.
Control	Gets a value indicating whether the CTRL key was pressed.
KeyCode	Gets a value which specifies the key that was pressed.
KeyValue	Gets the escape sequence mapped to the special key.
Shift	Gets a value indicating whether the SHIFT key was pressed.

Remarks

This event is generated when the user presses a special key while the emulation window has focus, and that key is mapped to a string using the KeyMap array.

See Also

Terminal Class | SocketTools Namespace

Terminal.KeyMappedEventArgs Class

Provides data for the KeyMapped event.

For a list of all members of this type, see Terminal.KeyMappedEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.Terminal.KeyMappedEventArgs

[Visual Basic]

```
Public Class Terminal.KeyMappedEventArgs
    Inherits EventArgs
```

[C#]

```
public class Terminal.KeyMappedEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The **KeyMapped** event is generated when the user presses a special key while the emulation window has focus, and that key is mapped to a string using the KeyMap array.

Requirements

Namespace: SocketTools

Assembly: SocketTools.Terminal (in SocketTools.Terminal.dll)

See Also

Terminal.KeyMappedEventArgs Members | SocketTools Namespace

Terminal.KeyMappedEventArgs Members

Terminal.KeyMappedEventArgs overview

Public Instance Constructors

Terminal.KeyMappedEventArgs Constructor	Initializes a new instance of the Terminal.KeyMappedEventArgs class.

Public Instance Properties

Alt	Gets a value indicating whether the ALT key was pressed.
Control	Gets a value indicating whether the CTRL key was pressed.
KeyCode	Gets a value which specifies the key that was pressed.
KeyValue	Gets the escape sequence mapped to the special key.
Shift	Gets a value indicating whether the SHIFT key was pressed.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

Terminal.KeyMappedEventArgs Class | SocketTools Namespace

Terminal.KeyMappedEventArgs Constructor

Initializes a new instance of the Terminal.KeyMappedEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public Terminal.KeyMappedEventArgs();
```

See Also

Terminal.KeyMappedEventArgs Class | SocketTools Namespace

Terminal.KeyMappedEventArgs Properties

The properties of the **Terminal.KeyMappedEventArgs** class are listed below. For a complete list of **Terminal.KeyMappedEventArgs** class members, see the Terminal.KeyMappedEventArgs Members topic.

Public Instance Properties

Alt	Gets a value indicating whether the ALT key was pressed.
Control	Gets a value indicating whether the CTRL key was pressed.
KeyCode	Gets a value which specifies the key that was pressed.
KeyValue	Gets the escape sequence mapped to the special key.
Shift	Gets a value indicating whether the SHIFT key was pressed.

See Also

Terminal.KeyMappedEventArgs Class | SocketTools Namespace

Terminal.KeyMappedEventArgs.Alt Property

Gets a value indicating whether the ALT key was pressed.

[Visual Basic]

```
Public ReadOnly Property Alt As Boolean
```

[C#]

```
public bool Alt {get;}
```

Property Value

A boolean value that is **true** if the ALT key was pressed; otherwise, **false**.

See Also

Terminal.KeyMappedEventArgs Class | SocketTools Namespace

Terminal.KeyMappedEventArgs.Control Property

Gets a value indicating whether the CTRL key was pressed.

[Visual Basic]

```
Public ReadOnly Property Control As Boolean
```

[C#]

```
public bool Control {get;}
```

Property Value

A boolean value that is **true** if the CTRL key was pressed; otherwise, **false**.

See Also

Terminal.KeyMappedEventArgs Class | SocketTools Namespace

Terminal.KeyMappedEventArgs.KeyCode Property

Gets a value which specifies the key that was pressed.

[Visual Basic]

```
Public ReadOnly Property KeyCode As TerminalKey
```

[C#]

```
public Terminal.TerminalKey KeyCode {get;}
```

Property Value

A TerminalKey enumeration value which specifies the key that was pressed.

See Also

Terminal.KeyMappedEventArgs Class | SocketTools Namespace

Terminal.KeyMappedEventArgs.KeyValue Property

Gets the escape sequence mapped to the special key.

[Visual Basic]

```
Public ReadOnly Property KeyValue As String
```

[C#]

```
public string KeyValue {get;}
```

Property Value

A string which specifies the escape sequence mapped to the key that was pressed.

See Also

Terminal.KeyMappedEventArgs Class | SocketTools Namespace

Terminal.KeyMappedEventArgs.Shift Property

Gets a value indicating whether the SHIFT key was pressed.

[Visual Basic]

```
Public ReadOnly Property Shift As Boolean
```

[C#]

```
public bool Shift {get;}
```

Property Value

A boolean value that is **true** if the SHIFT key was pressed; otherwise, **false**.

See Also

Terminal.KeyMappedEventArgs Class | SocketTools Namespace

Terminal.ErrorCode Enumeration

Specifies the error codes returned by the HttpClient class.

[Visual Basic]
```
Public Enum Terminal.ErrorCode
```

[C#]
```
public enum Terminal.ErrorCode
```

Remarks

The HttpClient class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorOutOfMemory	The specified file or directory does not exist.Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorInvalidBuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.

Requirements

Namespace: SocketTools

Assembly: SocketTools.Terminal (in SocketTools.Terminal.dll)

See Also

SocketTools Namespace

Terminal.TerminalCaret Enumeration

Specifies the caret styles supported by the Terminal control.

[Visual Basic]
```
Public Enum Terminal.TerminalCaret
```

[C#]
```
public enum Terminal.TerminalCaret
```

Members

Member Name	Description
None	No caret is displayed.
Underline	The caret is displayed as an underline.
Block	The caret is displayed as a block that is the full height of the character cell.

Requirements

Namespace: SocketTools

Assembly: SocketTools.Terminal (in SocketTools.Terminal.dll)

See Also

SocketTools Namespace

Terminal.TerminalAttributes Enumeration

Specifies the character cell attributes supported by the Terminal control.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum Terminal.TerminalAttributes
```

```
[C#]
[Flags]
public enum Terminal.TerminalAttributes
```

Members

Member Name	Description	Value
Normal	Normal, default attributes.	0
Reverse	Foreground and background cell colors are reversed.	1
Bold	The character is displayed using a higher intensity color.	2
Dim	The character is displayed using a lower intensity color.	4
Blink	This attribute is currently not supported.	8
Underline	The character is displayed with an underline.	16
Hidden	The character is stored in display memory, but not shown.	32
Protect	The character is protected and cannot be cleared.	64

Requirements

Namespace: SocketTools

Assembly: SocketTools.Terminal (in SocketTools.Terminal.dll)

See Also

SocketTools Namespace

Terminal.TerminalColor Enumeration

Specifies the colors supported by the Terminal control.

[Visual Basic]
```
Public Enum Terminal.TerminalColor
```

[C#]
```
public enum Terminal.TerminalColor
```

Remarks

A standard ANSI color terminal supports eight standard colors (0-7). To select a foreground color, you add 30 to the color index and pass that value as a parameter to the SGR (select graphic rendition) escape sequence. To select a background color, you add 40 to the color index. For example, to set the current foreground color to white and the background color to blue, you could send the following escape sequence:

```
ESC [ 37;44 m
```

Note that if you wanted to set the foreground color to a bold version of standard yellow, you would first set the bold attribute, and then use the index value of 3, such as:

```
ESC [ 1;33m
```

Changing the value of the ColorMap array allows the application to make selective changes to the actual RGB color value that is used when a color attribute is set. Note that changes to the color map will only affect new characters as they are displayed, not any previously displayed characters

Members

Member Name	Description
Black	Color value RGB(0,0,0)
Red	Color value RGB(160,0,0)
Green	Color value RGB(0,160,0)
Yellow	Color value RGB(160,160,0)
Blue	Color value RGB(0,0,160)
Magenta	Color value RGB(160,0,160)
Cyan	Color value RGB(0,160,160)
White	Color value RGB(224,224,224)
Gray	Color value RGB(192,192,192)
LtRed	Color value RGB(255,128,128)
LtGreen	Color value RGB(144,238,144)
LtYellow	Color value RGB(255,255,192)
LtBlue	Color value RGB(173,216,230)
LtMagenta	Color value RGB(255,192,255)
LtCyan	Color value RGB(224,255,255)

HiWhite	Color value RGB(255,255,255)

Requirements

Namespace: SocketTools

Assembly: SocketTools.Terminal (in SocketTools.Terminal.dll)

See Also

SocketTools Namespace

Terminal.TerminalEmulation Enumeration

Specifies the terminal emulation types supported by the Terminal class.

[Visual Basic]
```
Public Enum Terminal.TerminalEmulation
```

[C#]
```
public enum Terminal.TerminalEmulation
```

Members

Member Name	Description
None	The virtual display does not emulate any specific terminal type, and does not process any escape sequences.
ANSI	The virtual display will process ANSI escape sequences. The default keymap for an ANSI console is loaded. This is the default value.
VT100	The virtual display will process DEC VT100 escape sequences. The default keymap for a VT100 terminal is loaded.
VT220	The virtual display will process DEC VT220 escape sequences. The default keymap for a VT220 terminal is loaded.

Requirements

Namespace: SocketTools

Assembly: SocketTools.Terminal (in SocketTools.Terminal.dll)

See Also

SocketTools Namespace

Terminal.TerminalKey Enumeration

Specifies the special function keys which may be mapped by the Terminal control.

[Visual Basic]

```
Public Enum Terminal.TerminalKey
```

[C#]

```
public enum Terminal.TerminalKey
```

Remarks

This **TerminalKey** enumeration lists the constants which can be used as the index value with the KeyMap array to map control or escape sequences to certain keys. These values are also used by the KeyMapped event when the user presses a mapped key.

Members

Member Name	Description
F1	The F1 function key.
F2	The F2 function key.
F3	The F3 function key.
F4	The F4 function key.
F5	The F5 function key.
F6	The F6 function key.
F7	The F7 function key.
F8	The F8 function key.
F9	The F9 function key.
F10	The F10 function key.
F11	The F11 function key.
F12	The F12 function key.
ShiftF1	The Shift+F1 function key.
ShiftF2	The Shift+F2 function key.
ShiftF3	The Shift+F3 function key.
ShiftF4	The Shift+F4 function key.
ShiftF5	The Shift+F5 function key.
ShiftF6	The Shift+F6 function key.
ShiftF7	The Shift+F7 function key.
ShiftF8	The Shift+F8 function key.
ShiftF9	The Shift+F9 function key.
ShiftF10	The Shift+F10 function key.
ShiftF11	The Shift+F11 function key.

ShiftF12	The Shift+F12 function key.
Enter	The enter or return key.
Erase	The backspace or erase key.
Up	The cursor up key.
Down	The cursor down key.
Left	The cursor left key.
Right	The cursor right key.
Insert	The insert key.
Delete	The delete key.
Home	The home cursor key.
End	The end cursor key.
PageUp	The page up key.
PageDown	The page down key.
ArrowUp	The up arrow key.
ArrowDown	The down arrow key.
ArrowLeft	The left arrow key.
ArrowRight	The right arrow key.
KeypadEnter	The keypad enter key.
Keypad0	The keypad 0 key.
Keypad1	The keypad 1 key.
Keypad2	The keypad 2 key.
Keypad3	The keypad 3 key.
Keypad4	The keypad 4 key.
Keypad5	The keypad 5 key.
Keypad6	The keypad 6 key.
Keypad7	The keypad 7 key.
Keypad8	The keypad 8 key.
Keypad9	The keypad 9 key.

Requirements

Namespace: SocketTools

Assembly: SocketTools.Terminal (in SocketTools.Terminal.dll)

See Also

SocketTools Namespace

Terminal.TerminalNewLine Enumeration

Specifies the end-of-line character sequences supported by the Terminal control.

[Visual Basic]
```
Public Enum Terminal.TerminalNewLine
```

[C#]
```
public enum Terminal.TerminalNewLine
```

Members

Member Name	Description
CRLF	A carriage return positions the cursor to the first column, and a linefeed advances the cursor to the next row, scrolling the display if necessary. This is the default value.
CR	A carriage return positions the cursor to the first column and advances to the next row, scrolling the display if necessary.
LF	A linefeed positions the cursor to the first column and advances to the next row, scrolling the display if necessary.

Requirements

Namespace: SocketTools

Assembly: SocketTools.Terminal (in SocketTools.Terminal.dll)

See Also

SocketTools Namespace

Terminal.TerminalScrollBars Enumeration

Specifies the scrollbar modes supported by the Terminal control.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum Terminal.TerminalScrollBars
```

[C#]
```
[Flags]
public enum Terminal.TerminalScrollBars
```

Remarks

Scrollbars are only displayed if needed. If the emulation window is large enough to display all of the columns and rows, no scrollbars will be drawn even if they are enabled.

Members

Member Name	Description	Value
None	Scrollbars are not displayed.	0
Horizontal	A horizontal scrollbar is displayed if necessary.	1
Vertical	A vertical scrollbar is displayed if necessary.	2
Both	Both a horizontal vertical scrollbars are displayed if necessary.	3

Requirements

Namespace: SocketTools

Assembly: SocketTools.Terminal (in SocketTools.Terminal.dll)

See Also

SocketTools Namespace

Terminal.TerminalSelection Enumeration

The selection options supported by the Terminal control.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum Terminal.TerminalSelection
```

[C#]
```
[Flags]
public enum Terminal.TerminalSelection
```

Members

Member Name	Description	Value
Default	The default selection option. If there is a region of the display already selected, it will be cleared and the new region is selected. The selected text is buffered and can be accessed using the **SelText** property.	0
Clipboard	Copy the selected text to the clipboard. If this option is not specified, the selected text is buffered and may be accessed using the **SelText** property.	1
NoRefresh	The display is not refreshed when the region is selected. This is useful if the application is going to be selecting multiple regions of the display, or combining more than one region, in order to minimize output to the window.	4096
NoBuffer	Do not buffer the text in the selected region of the display. The display will show any text as being selected, but it will not be available to the application. This can be useful if the application is going to select multiple regions and combine them.	8192
Combine	If there is already a region of the display that has been selected, the new region is combined with the previous region, selecting all of the text.	16384

Requirements

Namespace: SocketTools

Assembly: SocketTools.Terminal (in SocketTools.Terminal.dll)

See Also

SocketTools Namespace

Terminal.KeyMappedEventHandler Delegate

Represents the method that will handle the KeyMapped event

[Visual Basic]

```
Public Delegate Sub Terminal.KeyMappedEventHandler( _
   ByVal sender As Object, _
   ByVal e As KeyMappedEventArgs _
)
```

[C#]

```
public delegate void Terminal.KeyMappedEventHandler(
      object sender,
      KeyMappedEventArgs e
   );
```

Parameters

sender
: The source of the event.

e
: A KeyMappedEventArgs that contains the event data.

Remarks

When you create an **KeyMappedEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **KeyMappedEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.Terminal (in SocketTools.Terminal.dll)

See Also

SocketTools Namespace

Terminal.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see Terminal.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.Terminal.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class Terminal.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class Terminal.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the Terminal class.

Example

```
<Assembly: SocketTools.Terminal.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.Terminal.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.Terminal (in SocketTools.Terminal.dll)

See Also

Terminal.RuntimeLicenseAttribute Members | SocketTools Namespace

Terminal.RuntimeLicenseAttribute Members

Terminal.RuntimeLicenseAttribute overview

Public Instance Constructors

Terminal.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

Terminal.RuntimeLicenseAttribute Class | SocketTools Namespace

Terminal.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public Terminal.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the Terminal class.

See Also

Terminal.RuntimeLicenseAttribute Class | SocketTools Namespace

Terminal.RuntimeLicenseAttribute Properties

The properties of the **Terminal.RuntimeLicenseAttribute** class are listed below. For a complete list of **Terminal.RuntimeLicenseAttribute** class members, see the Terminal.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

Terminal.RuntimeLicenseAttribute Class | SocketTools Namespace

Terminal.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

Terminal.RuntimeLicenseAttribute Class | SocketTools Namespace

TerminalEmulationException Class

The exception that is thrown when an error occurs.

For a list of all members of this type, see TerminalEmulationException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.TerminalEmulationException

[Visual Basic]

```
Public Class TerminalEmulationException
    Inherits ApplicationException
```

[C#]

```
public class TerminalEmulationException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A TerminalEmulationException is thrown by the Terminal control when an error occurs.

The default constructor for the TerminalEmulationException class sets the **ErrorCode** property to the socket error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.Terminal (in SocketTools.Terminal.dll)

See Also

TerminalEmulationException Members | SocketTools Namespace

TerminalEmulationException Members

TerminalEmulationException overview

Public Instance Constructors

TerminalEmulationException	Overloaded. Initializes a new instance of the TerminalEmulationException class.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

TerminalEmulationException Class | SocketTools Namespace

TerminalEmulationException Constructor

Initializes a new instance of the TerminalEmulationException class.

Overload List

Initializes a new instance of the TerminalEmulationException class.

public TerminalEmulationException();

Initializes a new instance of the TerminalEmulationException class with the specified error code.

public TerminalEmulationException(int);

Initializes a new instance of the TerminalEmulationException class with the specified error message.

public TerminalEmulationException(string);

Initializes a new instance of the TerminalEmulationException class with the specified error message and a reference to the inner exception that is the cause of this exception.

public TerminalEmulationException(string,Exception);

See Also

TerminalEmulationException Class | SocketTools Namespace

TerminalEmulationException Constructor ()

Initializes a new instance of the TerminalEmulationException class.

[Visual Basic]

```
Overloads Public Sub New()
```

[C#]

```
public TerminalEmulationException();
```

Return Value

The ctor constructor sets the **ErrorCode** property to the last socket error that occurred. For more information about the errors that may occur, refer to the Terminal.ErrorCode enumeration.

See Also

TerminalEmulationException Class | SocketTools Namespace | TerminalEmulationException Constructor Overload List

TerminalEmulationException Constructor (String)

Initializes a new instance of the TerminalEmulationException class with the specified error message.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]

```
public TerminalEmulationException(
   string message
);
```

Parameters

message

The error message that explains the reason for the exception.

Return Value

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

TerminalEmulationException Class | SocketTools Namespace | TerminalEmulationException Constructor Overload List

TerminalEmulationException Constructor (String, Exception)

Initializes a new instance of the TerminalEmulationException class with the specified error message and a reference to the inner exception that is the cause of this exception.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal innerException As Exception _
)
```

[C#]

```
public TerminalEmulationException(
   string message,
   Exception innerException
);
```

Parameters

message

The error message that explains the reason for the exception.

innerException

The exception that is the cause of the current exception. If the ***innerException*** parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

TerminalEmulationException Class | SocketTools Namespace | TerminalEmulationException Constructor Overload List

TerminalEmulationException Constructor (Int32)

Initializes a new instance of the TerminalEmulationException class with the specified error code.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

[C#]

```
public TerminalEmulationException(
   int code
);
```

Parameters

code

An integer value which specifies an error code.

Return Value

This constructor sets the **ErrorCode** property to the specified error code. For more information about the errors that may occur, refer to the Terminal.ErrorCode enumeration.

See Also

TerminalEmulationException Class | SocketTools Namespace | TerminalEmulationException Constructor Overload List

TerminalEmulationException Properties

The properties of the **TerminalEmulationException** class are listed below. For a complete list of **TerminalEmulationException** class members, see the TerminalEmulationException Members topic.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

TerminalEmulationException Class | SocketTools Namespace

TerminalEmulationException.ErrorCode Property

Gets a value which specifies the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property ErrorCode As ErrorCode
```

[C#]

```
public Terminal.ErrorCode ErrorCode {get;}
```

Property Value

Returns a Terminal.ErrorCode enumeration value which specifies the error code

Remarks

The **ErrorCode** property returns the error code that specifies the cause of the exception.

The default constructor for the TerminalEmulationException class sets the error code to the last network error that occurred. For more information about the errors that may occur, refer to the Terminal.ErrorCode enumeration.

See Also

TerminalEmulationException Class | SocketTools Namespace

TerminalEmulationException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]
```
Overrides Public ReadOnly Property Message As String
```

[C#]
```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

TerminalEmulationException Class | SocketTools Namespace

TerminalEmulationException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception.

See Also

TerminalEmulationException Class | SocketTools Namespace

TerminalEmulationException Methods

The methods of the **TerminalEmulationException** class are listed below. For a complete list of **TerminalEmulationException** class members, see the TerminalEmulationException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

TerminalEmulationException Class | SocketTools Namespace

TerminalEmulationException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]

```
Overrides Public Function ToString() As String
```

[C#]

```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

TerminalEmulationException Class | SocketTools Namespace

TextMessage Class

Send text messages to a mobile communications device using a gateway service.

For a list of all members of this type, see TextMessage Members.

System.Object
 SocketTools.TextMessage

[Visual Basic]

```
Public Class TextMessage
    Implements IDisposable
```

[C#]

```
public class TextMessage : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

Short Message Service (SMS) is a text messaging service used by mobile communication devices to exchange brief text messages. Most service providers also provide gateway servers that can be used to send messages to a wireless device on their network using standard email protocols. The **SocketTools.TextMessage** class provides methods that can be used to determine the provider associated with a specific telephone number and send a text message to the device using the provider's mail gateway.

This class has been designed to assist developers in sending text message notifications as part of their application. For example, it can be used to enable your software to automatically send notifications when a specific event occurs, such as an error condition. This class is not designed to be used with software that will send out a large number of text messages to many users, and there are limitations on the number of messages that may be sent to different phone numbers over a short period of time. Because many recipients must pay a fee for each text message they receive, text messages should only be sent to those who explicitly request them.

Note: This component only supports service providers in North America and cannot be used to send text messages to mobile devices that use providers outside of the United States and Canada. Some service providers may prevent messages from being sent through their gateway to a user that does not have unlimited text messaging as part of their service agreement.

Requirements

Namespace: SocketTools

Assembly: SocketTools.TextMessage (in SocketTools.TextMessage.dll)

See Also

TextMessage Members | SocketTools Namespace

TextMessage Members

TextMessage overview

Public Instance Constructors

TextMessage Constructor	Initializes a new instance of the TextMessage class.

Public Instance Properties

Account	Gets and sets a value which specifies an account name used to authenticate a client session.
AuthType	Gets and sets the type of client authentication that should be used.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
Message	Gets and sets the current message text.
Password	Gets and sets the password used to authenticate the session.
PhoneNumber	Gets and sets the phone number for the mobile device.
Provider	Gets and sets the name of the preferred wireless service provider.
Providers	Gets a collection of known wireless service providers.
Relay	Enable or disable the use of an intermediate relay server to send messages.
Secure	Enable or disable secure connections to the server.
Sender	Gets and sets the value that identifies the sender of the message.
ServerName	Gets and sets the name of the server that is used to send messages.
ServerPort	Gets and sets the port number used to establish a connection.
ServiceType	Gets and sets the type of service used to send messages.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout

	period in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
Urgent	Enable or disable the option that flags a message as urgent.
UserName	Gets and sets the user name or ID used to authenticate the session.
Version	Gets a value which returns the current version of the TextMessage class library.

Public Instance Methods

Dispose	Overloaded. Releases all resources used by TextMessage.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetAddress	Get the email address associated with the specified phone number.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetProvider	Get the name of the wireless service provider associated with a phone number.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the TextMessage class.
Reset	Reset the internal state of the object, resetting all properties to their default values.
SendMessage	Overloaded. Send a text message to the specified mobile device.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.

Public Instance Events

OnError	Occurs when an client operation fails.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the TextMessage class and optionally

	releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

TextMessage Class | SocketTools Namespace

TextMessage Constructor

Initializes a new instance of the TextMessage class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public TextMessage();
```

See Also

TextMessage Class | SocketTools Namespace

TextMessage Properties

The properties of the **TextMessage** class are listed below. For a complete list of **TextMessage** class members, see the TextMessage Members topic.

Public Instance Properties

Account	Gets and sets a value which specifies an account name used to authenticate a client session.
AuthType	Gets and sets the type of client authentication that should be used.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
Message	Gets and sets the current message text.
Password	Gets and sets the password used to authenticate the session.
PhoneNumber	Gets and sets the phone number for the mobile device.
Provider	Gets and sets the name of the preferred wireless service provider.
Providers	Gets a collection of known wireless service providers.
Relay	Enable or disable the use of an intermediate relay server to send messages.
Secure	Enable or disable secure connections to the server.
Sender	Gets and sets the value that identifies the sender of the message.
ServerName	Gets and sets the name of the server that is used to send messages.
ServerPort	Gets and sets the port number used to establish a connection.
ServiceType	Gets and sets the type of service used to send messages.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
Trace	Gets and sets a value which indicates if network

	function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
Urgent	Enable or disable the option that flags a message as urgent.
UserName	Gets and sets the user name or ID used to authenticate the session.
Version	Gets a value which returns the current version of the TextMessage class library.

See Also

TextMessage Class | SocketTools Namespace

TextMessage.Account Property

Gets and sets a value which specifies an account name used to authenticate a client session.

[Visual Basic]

```
Public Property Account As String
```

[C#]

```
public string Account {get; set;}
```

Property Value

A string which specifies the account name.

Remarks

This property is provided for future expansion when additional gateway service types and authentication methods are added to this component. This **Account** property is not used when a message is sent using an SMTP gateway.

See Also

TextMessage Class | SocketTools Namespace

TextMessage.AuthType Property

Gets and sets the type of client authentication that should be used.

[Visual Basic]

```
Public Property AuthType As AuthTypes
```

[C#]

```
public TextMessage.AuthTypes AuthType {get; set;}
```

Property Value

A TextMessage.AuthTypes enumeration that specifies the authentication method.

Remarks

This property is provided for future expansion when additional gateway service types and authentication methods are added to this component. Applications should not change the value of the **AuthType** property.

See Also

TextMessage Class | SocketTools Namespace | TextMessage.AuthTypes

TextMessage.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

TextMessage.ErrorEventArgs Class | SocketTools Namespace | Error Property

TextMessage.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public TextMessage.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

TextMessage Class | SocketTools Namespace

TextMessage.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

TextMessage Class | SocketTools Namespace

TextMessage.Message Property

Gets and sets the current message text.

[Visual Basic]

```
Public Property Message As String
```

[C#]

```
public string Message {get; set;}
```

Property Value

A string that contains the current text message.

Remarks

This property returns the current message text. Changing the value of this property will change the message text sent by the **SendMessage** method if no message is explicitly specified. In most cases, a message should not exceed 160 characters in length, although some service providers may accept longer messages. If a message exceeds the maximum number of characters accepted by a service provider, the message may be ignored or it may be split into multiple messages.

See Also

TextMessage Class | SocketTools Namespace | PhoneNumber Property | Sender Property | SendMessage Method

TextMessage.Password Property

Gets and sets the password used to authenticate the session.

[Visual Basic]

```
Public Property Password As String
```

[C#]

```
public string Password {get; set;}
```

Property Value

A string that specifies the current password used to authenticate the client session.

Remarks

This property is used to authenticate the current client session. If the message is being sent using SMTP, this value is used to specify a password when connecting to the mail server . If no authentication is required, this property may be an empty string. Note that some service providers may use terminology other than "password" with their documentation; in that case, this property will always specify the second of a pair of authentication tokens sent to the server to identify the client.

See Also

TextMessage Class | SocketTools Namespace | Relay Property | ServerName Property | ServerPort Property | UserName Property

TextMessage.PhoneNumber Property

Gets and sets the phone number for the mobile device.

[Visual Basic]

```
Public Property PhoneNumber As String
```

[C#]

```
public string PhoneNumber {get; set;}
```

Property Value

A string which specifies the phone number for the mobile device.

Remarks

The **PhoneNumber** property returns the current phone number. Changing the value of this property will change the default phone number the **SendMessage** method will use when sending a message. This can be a standard E.164 formatted phone number or an unformatted number. Any extraneous whitespace, punctuation or other non-numeric characters in the string will be ignored.

See Also

TextMessage Class | SocketTools Namespace | Message Property | Relay Property | Sender Property | SendMessage Method

TextMessage.Provider Property

Gets and sets the name of the preferred wireless service provider.

[Visual Basic]

```
Public Property Provider As String
```

[C#]

```
public string Provider {get; set;}
```

Property Value

A string which specifies the preferred wireless service provider associated with the mobile device.

Remarks

This property returns the preferred wireless service provider associated with the current phone number. Changing the value of this property will change the preferred wireless service provider. If this property is an empty string, the default provider assigned to the recipient's phone number will be used. This property is only used with messages sent using SMTP and is ignored for other message services.

In the United States and Canada, most wireless common carriers are required to provide wireless number portability (WNP) which allows a customer to continue to use their current phone number even if they switch to another service provider. This can result in a situation where a specific phone number is shown as allocated to one provider, but in actuality that user has switched to a different provider. For example, a user may have originally purchased a phone and service with AT&T and then later switched to Verizon, but decided to keep their phone number. In this case, if Verizon was not specified as the preferred provider, the library would attempt to send the message to the AT&T gateway, since that was the original provider who allocated the phone number.

For most applications, the correct way to handle the situation in which a user may have switched to a different service provider is to allow them to select a preferred provider in your user interface. For example, you could display a drop-down list of available providers for them to select from, populated using the **Providers** property. If they select a preferred provider, then you would assign that value to this property. If they choose to use the default provider, set this property to an empty string.

See Also

TextMessage Class | SocketTools Namespace | Providers Property | SendMessage Method

TextMessage.Providers Property

Gets a collection of known wireless service providers.

[Visual Basic]

```
Public Property Providers As ArrayList
```

[C#]

```
public System.Collections.ArrayList Providers {get; set;}
```

Property Value

A collection of strings that specify the names of known wireless service providers.

Remarks

This property will return a read-only collection of known wireless service providers in the United States and Canada. Typically this collection would be used to populate a UI control such as a ListBox or ComboBox to provide the user with a list of service providers to choose from.

Attempting to assign a value to this property will cause a **System.NotSupportedException** exception to be raised.

See Also

TextMessage Class | SocketTools Namespace | Provider Property | GetProvider Method

TextMessage.Relay Property

Enable or disable the use of an intermediate relay server to send messages.

[Visual Basic]

```
Public Property Relay As Boolean
```

[C#]

```
public bool Relay {get; set;}
```

Property Value

A boolean value which specifies if a relay mail server will be used to send the text message. A value of **true** specifies that a relay server will be used, while a value of **false** specifies that the message will be sent directly to the wireless service provider's gateway.

Remarks

The **Relay** property is used to determine if the class will send the message directly to the wireless service provider's gateway server, or if the message will be relayed through another mail server. The default value for this property is **false**. Setting this property to **true** will cause the control to connect to the mail server specified by the **ServerName** property, authenticate the client session if necessary and then submit the message.

When a text message is sent using the SMTP service, the default action is to attempt to connect directly to the wireless service provider's gateway server. However, many residential Internet service providers (ISPs) do not permit their customers to connect to third-party mail servers and will block the outbound connection. Some wireless service providers may also reject messages that originate from residential IP addresses.

To resolve this issue, the developer should allow the user to specify an alternate mail server that will relay the message to the wireless service provider. For residential users, this will typically be the mail server provided by their ISP. For business users, this will usually be their corporate mail server. The **ServerName** and **ServerPort** properties are used to identify the relay server, and the **UserName** and **Password** properties provide the credentials to authenticate the client session.

See Also

TextMessage Class | SocketTools Namespace | Password Property | ServerName Property | ServerPort Property | UserName Property

TextMessage.Secure Property

Enable or disable secure connections to the server.

[Visual Basic]

```
Public Property Secure As Boolean
```

[C#]

```
public bool Secure {get; set;}
```

Property Value

A boolean value which specifies if a secure connection should be established.

Remarks

The **Secure** property determines if a secure connection is established to the server. The default value for this property is **false**, which specifies that a standard connection to the server is used. To establish a secure connection, the application must set this property value to **true** prior to calling the **SendMessage** method.

This property is only used when sending a message through a relay server using the SMTP service. Messages that are sent directly to the wireless service provider's gateway do not use authentication and are not secure, regardless of the value of this property.

See Also

TextMessage Class | SocketTools Namespace | Relay Property | SendMessage Method

TextMessage.Sender Property

Gets and sets the value that identifies the sender of the message.

[Visual Basic]

```
Public Property Sender As String
```

[C#]

```
public string Sender {get; set;}
```

Property Value

A string value that identifies the person sending the text message.

Remarks

The **Sender** property returns the email address of the sender when the SMTP service is used to send the message. For other service types, this property typically specifies the phone number or shortcode associated with the sender. Assigning a value to this property will set the default sender when the **SendMessage** method is called.

See Also

TextMessage Class | SocketTools Namespace | PhoneNumber Property | SendMessage Method

TextMessage.ServerName Property

Gets and sets the name of the server that is used to send messages.

[Visual Basic]

```
Public Property ServerName As String
```

[C#]

```
public string ServerName {get; set;}
```

Property Value

A string that specifies the host name or IP address of a mail server.

Remarks

The **ServerName** property is used to specify an alternate server that is used to deliver messages. When the SMTP service is used, this property is used in conjunction with the **Relay** property to enable the relaying of messages through another mail server.

By default, the phone number is used to automatically determine the host name of the gateway server that is responsible for accepting messages for the mobile device. However, under some circumstances it may not be possible to send messages directly to the wireless service provider's gateway. For example, many Internet service providers (ISPs) require that customers relay all messages through their servers and block any attempt to establish a direct connection with another mail server. This property can be used to specify an alternate server that will be responsible for sending the message.

See Also

TextMessage Class | SocketTools Namespace | Relay Property | ServerPort Property | SendMessage Method

TextMessage.ServerPort Property

Gets and sets the port number used to establish a connection.

[Visual Basic]

```
Public Property ServerPort As Integer
```

[C#]

```
public int ServerPort {get; set;}
```

Property Value

An integer value that specifies the mail server port number.

Remarks

The **ServerPort** property defines the port number which is used to establish a connection with the server. This property is used in conjunction with the **ServerName** property to specify an alternate server which is responsible for delivering messages.

See Also

TextMessage Class | SocketTools Namespace | Relay Property | ServerName Property | SendMessage Method

TextMessage.ServiceType Property

Gets and sets the type of service used to send messages.

[Visual Basic]
```
Public Property ServiceType As ServiceTypes
```

[C#]
```
public TextMessage.ServiceTypes ServiceType {get; set;}
```

Property Value

A SocketTools.ServiceTypes enumeration that specifies the type of service.

Remarks

This property is provided for future expansion when additional gateway service types and authentication methods are added to this component. Applications should not change the value of the **ServiceType** property.

See Also

TextMessage Class | SocketTools Namespace | TextMessage.ServiceTypes

TextMessage.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]

```
Public Property ThrowError As Boolean
```

[C#]

```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

TextMessage Class | SocketTools Namespace

TextMessage.Timeout Property

Gets and sets a value which specifies a timeout period in seconds.

[Visual Basic]

```
Public Property Timeout As Integer
```

[C#]

```
public int Timeout {get; set;}
```

Property Value

An integer value which specifies a timeout period in seconds.

Remarks

Setting the **Timeout** property specifies the number of seconds until a blocking operation fails and the method returns to the caller. This includes the amount of time the control will spend attempting to connect to a remote host, and if the connection is not established within the given time period, the connection attempt will fail.

See Also

TextMessage Class | SocketTools Namespace

TextMessage.Trace Property

Gets and sets a value which indicates if network function logging is enabled.

[Visual Basic]

```
Public Property Trace As Boolean
```

[C#]

```
public bool Trace {get; set;}
```

Property Value

Returns **true** if network function tracing is enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Trace** property is used to enable (or disable) the tracing of network function calls. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the SocketTools networking classes and is enabled or disabled for an entire process. This means that once trace logging is enabled for a given component, all of the function calls made by the process using any of the SocketTools classes will be logged. For example, if you have an application using both the File Transfer Protocol and Post Office Protocol classes, and you set the **Trace** property to **true**, function calls made by both classes will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all classes used by the process.

If trace logging is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the logfile is fairly large. Since logfiles can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

When redistributing your application, make sure that you include the **SocketTools10.TraceLog.dll** module with your installation. If this library is not present, then no trace output will be generated and the value of the **Trace** property will be ignored. Only those function calls made by the SocketTools networking classes will be logged. Calls made directly to the Windows Sockets API, or calls made by other classes, will not be logged.

See Also

TextMessage Class | SocketTools Namespace

TextMessage.TraceFile Property

Gets and sets a value which specifies the name of the logfile.

[Visual Basic]

```
Public Property TraceFile As String
```

[C#]

```
public string TraceFile {get; set;}
```

Property Value

A string which specifies the name of the file.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named **SocketTools.log** is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFile** property should be set to the same value for each instance. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
 MyApp INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0 MyApp WRN:
connect(46, 192.0.0.1:1234, 16) returned -1 [10035] MyApp ERR: accept(46,
NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the control is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Note that if the specified file cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

TextMessage Class | SocketTools Namespace

TextMessage.TraceFlags Property

Gets and sets a value which specifies the client function tracing flags.

[Visual Basic]
```
Public Property TraceFlags As TraceOptions
```

[C#]
```
public TextMessage.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration which specifies the amount of detail written to the trace logfile.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Because network function tracing is enabled per-process, the trace flags are shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFlags** property should be set to the same value for each instance. Changing the trace flags for any one instance of the class will affect the logging performed for all SocketTools classes used by the application.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being written and the error **errorOperationWouldBlock** occurs, a warning is generated because the application simply needs to attempt to write the data at a later time.

See Also

TextMessage Class | SocketTools Namespace

TextMessage.Urgent Property

Enable or disable the option that flags a message as urgent.

[Visual Basic]

```
Public Property Urgent As Boolean
```

[C#]

```
public bool Urgent {get; set;}
```

Property Value

A boolean value which specifies if the message should be flagged as urgent.

Remarks

The **Urgent** property specifies whether a message will be flagged as urgent or not. If this property is set to **true**, the message will sent with a high priority. Note that this does not guarantee the message will be received any differently than a standard text message. Each wireless service provider may handle urgent messages differently, and some providers may simply ignore the message priority. By default, this property is **false**.

See Also

TextMessage Class | SocketTools Namespace

TextMessage.UserName Property

Gets and sets the user name or ID used to authenticate the session.

[Visual Basic]

```
Public Property UserName As String
```

[C#]

```
public string UserName {get; set;}
```

Property Value

A string that specifies the current username used to authenticate the client session.

Remarks

The **UserName** property is used to authenticate the current session. If the message is being sent using SMTP, this property is used to specify a user name when connecting to the mail server . If no authentication is required, this property may be an empty string. Note that some service providers may use terminology other than "username" with their documentation; in that case, this property will always specify the first of a pair of authentication tokens sent to the server to identify the client.

See Also

TextMessage Class | SocketTools Namespace | Password Property

TextMessage.Version Property

Gets a value which returns the current version of the TextMessage class library.

[Visual Basic]
```
Public ReadOnly Property Version As String
```

[C#]
```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the TextMessage class library. This value can be used by an application for validation and debugging purposes.

See Also

TextMessage Class | SocketTools Namespace

TextMessage Methods

The methods of the **TextMessage** class are listed below. For a complete list of **TextMessage** class members, see the TextMessage Members topic.

Public Instance Methods

Dispose	Overloaded. Releases all resources used by TextMessage.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetAddress	Get the email address associated with the specified phone number.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetProvider	Get the name of the wireless service provider associated with a phone number.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the TextMessage class.
Reset	Reset the internal state of the object, resetting all properties to their default values.
SendMessage	Overloaded. Send a text message to the specified mobile device.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the TextMessage class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

TextMessage Class | SocketTools Namespace

TextMessage.Dispose Method

Releases all resources used by TextMessage.

Overload List

Releases all resources used by TextMessage.

public void Dispose();

Releases the unmanaged resources allocated by the TextMessage class and optionally releases the managed resources.

protected virtual void Dispose(bool);

See Also

TextMessage Class | SocketTools Namespace

TextMessage.Dispose Method ()

Releases all resources used by TextMessage.

[Visual Basic]
```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]
```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

TextMessage Class | SocketTools Namespace | TextMessage.Dispose Overload List

TextMessage.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the TextMessage class and optionally releases the managed resources.

[Visual Basic]

```
Overridable Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]

```
protected virtual void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **TextMessage** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

TextMessage Class | SocketTools Namespace | TextMessage.Dispose Overload List

TextMessage.Finalize Method

Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.

[Visual Basic]

```
Overrides Protected Sub Finalize()
```

[C#]

```
protected override void Finalize();
```

See Also

TextMessage Class | SocketTools Namespace

TextMessage.GetAddress Method

Get the email address associated with the specified phone number.

[Visual Basic]

```
Public Function GetAddress( _
   ByVal phoneNumber As String, _
   ByRef mailAddress As String _
) As Boolean
```

[C#]

```
public bool GetAddress(
   string phoneNumber,
   ref string mailAddress
);
```

Parameters

phoneNumber

A string value which specifies the phone number of the mobile device. This can be a standard E.164 formatted number or an unformatted number. Any extraneous whitespace, punctuation or other non-numeric characters in the string will be ignored.

mailAddress

A string that will contain the email address associated with the specified phone number when the method returns. A mail message sent to this address will be forwarded to the mobile device as a text message. This parameter must be passed by reference.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetAddress** method can be used to determine if a phone number is associated with a mobile device and obtain the email address for the wireless service provider's gateway. This is done by sending an query to a server that will check the phone number against a database of known providers and the phone numbers that have been allocated for wireless devices.

This method sends an HTTP query to the server **api.sockettools.com** to obtain information about the phone number and wireless service provider. This requires that the local system can establish a standard network connection over port 80. If the client cannot connect to the server, the method will fail and an appropriate error will be returned. The server imposes a limit on the maximum number of connections that can be established and the maximum number of requests that can be issued per minute. If this method is called multiple times over a short period, the control may also force the application to block briefly. Server responses are cached per session, so calling this method multiple times using the same phone number will not increase the request count.

See Also

TextMessage Class | SocketTools Namespace | Provider Property | Providers Property | GetProvider Method | SendMessage Method

TextMessage.GetProvider Method

Get the name of the wireless service provider associated with a phone number.

[Visual Basic]

```
Public Function GetProvider( _
   ByVal phoneNumber As String, _
   ByRef providerName As String _
) As Boolean
```

[C#]

```
public bool GetProvider(
   string phoneNumber,
   ref string providerName
);
```

Parameters

phoneNumber
A string value which specifies the phone number of the mobile device. This can be a standard E.164 formatted number or an unformatted number. Any extraneous whitespace, punctuation or other non-numeric characters in the string will be ignored.

providerName
A string that will contain the name of the wireless service provider associated with the specified phone number when the method returns. This parameter must be passed by reference.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **GetProvider** method can be used to determine if a phone number is associated with a mobile device and obtain the name of the service provider associated with the number. This is done by sending an query to a server that will check the phone number against a database of known providers and the phone numbers that have been allocated for wireless devices.

This method sends an HTTP query to the server **api.sockettools.com** to obtain information about the phone number and wireless service provider. This requires that the local system can establish a standard network connection over port 80. If the client cannot connect to the server, the method will fail and an appropriate error will be returned. The server imposes a limit on the maximum number of connections that can be established and the maximum number of requests that can be issued per minute. If this method is called multiple times over a short period, the control may also force the application to block briefly. Server responses are cached per session, so calling this method multiple times using the same phone number will not increase the request count.

See Also

TextMessage Class | SocketTools Namespace | Provider Property | Providers Property | GetAddress Method | SendMessage Method

TextMessage.Initialize Method

Initialize an instance of the TextMessage class.

Overload List

Initialize an instance of the TextMessage class.

public bool Initialize();

Initialize an instance of the TextMessage class.

public bool Initialize(string);

See Also

TextMessage Class | SocketTools Namespace | Uninitialize Method

TextMessage.Initialize Method ()

Initialize an instance of the TextMessage class.

[Visual Basic]
```
Overloads Public Function Initialize() As Boolean
```

[C#]
```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The **Initialize** method can be used to explicitly initialize an instance of the **TextMessage** class, loading the networking library and allocating resources for the current thread. This must be the first method that is called before you attempt to modify any property values or invoke other methods in this instance. Failure to initialize the class may result in in subsequent errors and/or cause an exception to be raised.

Each time the **Initialize** method is invoked, it increments an internal counter that keeps track of the number of times that it has been called by any thread in the current process. The **Uninitialize** method decrements this counter, and when the usage count drops to zero, the class will automatically unload the system libraries that it has dynamically loaded and will destroy the process heap that was allocated when the first instance of the class was created.

See Also

TextMessage Class | SocketTools Namespace | TextMessage.Initialize Overload List | Uninitialize Method

TextMessage.Initialize Method (String)

Initialize an instance of the TextMessage class.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The **Initialize** method can be used to explicitly initialize an instance of the **TextMessage** class, loading the networking library and allocating resources for the current thread. This must be the first method that is called before you attempt to modify any property values or invoke other methods in this instance. Failure to initialize the class may result in in subsequent errors and/or cause an exception to be raised.

Each time the **Initialize** method is invoked, it increments an internal counter that keeps track of the number of times that it has been called by any thread in the current process. The **Uninitialize** method decrements this counter, and when the usage count drops to zero, the class will automatically unload the system libraries that it has dynamically loaded and will destroy the process heap that was allocated when the first instance of the class was created.

Example

The following example shows how to use the Initialize method to initialize an instance of the class. This example assumes that the license key string has been defined in code.

```
SocketTools.TextMessage txtMessage = new SocketTools.TextMessage();

if (txtMessage.Initialize(strLicenseKey) == false)
{
 MessageBox.Show(txtMessage.LastErrorString, "Error",
 MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
 return;
}
```

```
Dim txtMessage As New SocketTools.TextMessage

If txtMessage.Initialize(strLicenseKey) = False Then
    MsgBox(txtMessage.LastErrorString, vbIconExclamation)
    Exit Sub
End If
```

See Also

TextMessage Class | SocketTools Namespace | TextMessage.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

TextMessage.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]
```
Public Sub Reset()
```

[C#]
```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. If a socket has been allocated, it will be released and any active connections will be terminated. All properties will be reset to their default values.

See Also

TextMessage Class | SocketTools Namespace

TextMessage.SendMessage Method

Send a text message to the specified mobile device.

Overload List

Send a text message to the specified mobile device.

public bool SendMessage();

Send a text message to the specified mobile device.

public bool SendMessage(string);

Send a text message to the specified mobile device.

public bool SendMessage(string,string,string);

See Also

TextMessage Class | SocketTools Namespace | Message Property | PhoneNumber Property | Relay Property | Sender Property | ServiceType Property

TextMessage.SendMessage Method ()

Send a text message to the specified mobile device.

[Visual Basic]

```
Overloads Public Function SendMessage() As Boolean
```

[C#]

```
public bool SendMessage();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SendMessage** method is used to send a text message to a mobile device. This class is designed to support multiple methods of sending text messages, with the **ServiceType** property determining how the message is sent.

The value of the **PhoneNumber** property is used to determine the phone number of the mobile device that the message will be sent to. The value of the **Sender** property is used to specify the sender of the message. The value of the **Message** property specifies the content of the message that is sent.

For more information, refer to the SendMessage method.

See Also

TextMessage Class | SocketTools Namespace | TextMessage.SendMessage Overload List | Message Property | PhoneNumber Property | Relay Property | Sender Property | ServiceType Property

TextMessage.SendMessage Method (String)

Send a text message to the specified mobile device.

[Visual Basic]

```
Overloads Public Function SendMessage( _
   ByVal phoneNumber As String _
) As Boolean
```

[C#]

```
public bool SendMessage(
   string phoneNumber
);
```

Parameters

phoneNumber

An string value which specifies the phone number of the mobile device. This can be a standard E.164 formatted number or an unformatted number. Any extraneous whitespace, punctuation or other non-numeric characters in the string will be ignored.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SendMessage** method is used to send a text message to a mobile device. This class is designed to support multiple methods of sending text messages, with the **ServiceType** property determining how the message is sent.

The value of the **Sender** property is used to specify the sender of the message. The value of the **Message** property specifies the content of the message that is sent.

For more information, refer to the SendMessage method.

See Also

TextMessage Class | SocketTools Namespace | TextMessage.SendMessage Overload List | Message Property | Relay Property | Sender Property | ServiceType Property

TextMessage.SendMessage Method (String, String, String)

Send a text message to the specified mobile device.

[Visual Basic]

```
Overloads Public Function SendMessage( _
   ByVal phoneNumber As String, _
   ByVal sender As String, _
   ByVal message As String _
) As Boolean
```

[C#]

```
public bool SendMessage(
   string phoneNumber,
   string sender,
   string message
);
```

Parameters

phoneNumber
An string value which specifies the phone number of the mobile device. This can be a standard E.164 formatted number or an unformatted number. Any extraneous whitespace, punctuation or other non-numeric characters in the string will be ignored.

sender
A string value that identifies the sender of the message. For messages being sent using SMTP, this should be a valid email address.

message
A string value that contains the text message that will be sent.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **SendMessage** method is used to send a text message to a mobile device. This control is designed to support multiple methods of sending text messages, with the **ServiceType** property determining how the message is sent:

ServiceTypes.serviceSmtp
This message service sends the message through the wireless service provider's mail gateway using the SMTP protocol. However, it is important to note that many of these gateways will not accept messages from a client that is connected to them using a residential Internet service provider. If the application is being run on a system that uses a residential provider, that service provider may also block outbound connections to all mail servers other than their own. These anti-spam measures typically require that most end-user applications specify a relay mail server rather than submitting the message directly to the wireless provider's gateway.

Because most wireless carriers in the United States and Canada must provide for wireless number portability, there is the possibility that the provider information returned may no longer correspond to the telephone number. It is recommended that you provide your end-user with the ability to specify an alternate preferred provider to use when sending the text message. For more information, refer to **Providers** properties.

This method sends an HTTP query to the server **api.sockettools.com** to obtain information about the phone number and wireless service provider. This requires that the local system can establish a standard network connection over port 80. If the client cannot connect to the server, the method will fail and an appropriate error will be returned. The server imposes a limit on the maximum number of connections that can be established and the maximum number of requests that can be issued per minute. If this method is called multiple times over a short period, the control may also force the application to block briefly. Server responses are cached per session, so calling this method multiple times using the same phone number will not increase the request count.

See Also

TextMessage Class | SocketTools Namespace | TextMessage.SendMessage Overload List | Providers Property | Relay Property | ServiceType Property

TextMessage.Uninitialize Method

Uninitialize the class library and release any resources allocated for the current thread.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method terminates any connection established by the control and resets the internal state of the class. This method is not typically used because any resources that have been allocated by an instance of the class will automatically be released when it is destroyed.

Each time the **Initialize** method is invoked, it increments an internal counter that keeps track of the number of times that it has been called by any thread in the current process. The **Uninitialize** method decrements this counter, and when the usage count drops to zero, the class will automatically unload the system libraries that it has dynamically loaded and will destroy the process heap that was allocated when the first instance of the class was created.

See Also

TextMessage Class | SocketTools Namespace | Initialize Method

TextMessage Events

The events of the **TextMessage** class are listed below. For a complete list of **TextMessage** class members, see the TextMessage Members topic.

Public Instance Events

⚡ OnError	Occurs when an client operation fails.

See Also

TextMessage Class | SocketTools Namespace

TextMessage.OnError Event

Occurs when an client operation fails.

[Visual Basic]

```
Public Event OnError As OnErrorEventHandler
```

[C#]

```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type TextMessage.ErrorEventArgs containing data related to this event. The following **TextMessage.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a client operation fails.

See Also

TextMessage Class | SocketTools Namespace | ErrorCode Enumeration | LastError Property | LastErrorString Property

TextMessage.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see TextMessage.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.TextMessage.ErrorEventArgs

[Visual Basic]

```
Public Class TextMessage.ErrorEventArgs
    Inherits EventArgs
```

[C#]

```
public class TextMessage.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.TextMessage (in SocketTools.TextMessage.dll)

See Also

TextMessage.ErrorEventArgs Members | SocketTools Namespace

TextMessage.ErrorEventArgs Members

TextMessage.ErrorEventArgs overview

Public Instance Constructors

TextMessage.ErrorEventArgs Constructor	Initializes a new instance of the TextMessage.ErrorEventArgs class.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

TextMessage.ErrorEventArgs Class | SocketTools Namespace

TextMessage.ErrorEventArgs Constructor

Initializes a new instance of the TextMessage.ErrorEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public TextMessage.ErrorEventArgs();
```

See Also

TextMessage.ErrorEventArgs Class | SocketTools Namespace

TextMessage.ErrorEventArgs Properties

The properties of the **TextMessage.ErrorEventArgs** class are listed below. For a complete list of **TextMessage.ErrorEventArgs** class members, see the TextMessage.ErrorEventArgs Members topic.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

TextMessage.ErrorEventArgs Class | SocketTools Namespace

TextMessage.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public TextMessage.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

TextMessage.ErrorEventArgs Class | SocketTools Namespace | Description Property

TextMessage.AuthTypes Enumeration

Specifies the authentication methods that the TextMessage class supports.

[Visual Basic]

```
Public Enum TextMessage.AuthTypes
```

[C#]

```
public enum TextMessage.AuthTypes
```

Members

Member Name	Description
authDefault	The default authentication method for this service type should be used. Most applications should use this value unless a service type provides multiple authentication methods.
authUserName	The service requires authentication using a username and password. This value can be used with an SMTP service that requires user authentication and is typically needed when using a mail server relay. For the **serviceSmtp** service, this is the default authentication method.

Requirements

Namespace: SocketTools

Assembly: SocketTools.TextMessage (in SocketTools.TextMessage.dll)

See Also

SocketTools Namespace | ServiceTypes Enumeration | AuthType Property (SocketTools.TextMessage)

TextMessage.ErrorCode Enumeration

Specifies the error codes returned by the TextMessage class.

[Visual Basic]
```
Public Enum TextMessage.ErrorCode
```

[C#]
```
public enum TextMessage.ErrorCode
```

Remarks

The TextMessage class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.
errorProxyAuthenticationRequired	Proxy authentication required.

errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.
errorUnsupportedMediaType	Request specified an unsupported media type.

errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.
errorNoMoreMailboxes	No more mailboxes exist on this server.

errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.
errorInvalidThreadHandle	The specified thread handle is no longer valid.

errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been defined.

errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or unrecognized.

errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.

Requirements

Namespace: SocketTools

Assembly: SocketTools.TextMessage (in SocketTools.TextMessage.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

TextMessage.ServiceTypes Enumeration

Specifies the gateway services that the TextMessage class supports.

[Visual Basic]
```
Public Enum TextMessage.ServiceTypes
```

[C#]
```
public enum TextMessage.ServiceTypes
```

Members

Member Name	Description
serviceSmtp	The text message will be sent through the mail gateway for the specified service provider. This service uses SMTP to submit the message for delivery, either directly to the server provider's mail gateway server or through a relay server. This is the default service type.

Requirements

Namespace: SocketTools

Assembly: SocketTools.TextMessage (in SocketTools.TextMessage.dll)

See Also

SocketTools Namespace | AuthTypes Enumeration | ServiceType Property (SocketTools.TextMessage)

TextMessage.TraceOptions Enumeration

Specifies the logging options that the TextMessage class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum TextMessage.TraceOptions
```

[C#]

```
[Flags]
public enum TextMessage.TraceOptions
```

Remarks

The TextMessage class uses the **TraceOptions** enumeration to specify what kind of debugging information is written to the trace logfile. These options are only meaningful when trace logging is enabled by setting the **Trace** property to **true**.

Members

Member Name	Description	Value
traceDefault	The default trace logging option. This is the same as specifying the **traceInfo** option.	0
traceInfo	All network function calls are written to the trace file. This is the default value.	0
traceError	Only those network function calls which fail are recorded in the trace file.	1
traceWarning	Only those network function calls which fail, or return values which indicate a warning, are recorded in the trace file.	2
traceHexDump	All network function calls are written to the trace file, plus all the data that is sent or received is displayed, in both ASCII and hexadecimal format.	4

Requirements

Namespace: SocketTools

Assembly: SocketTools.TextMessage (in SocketTools.TextMessage.dll)

See Also

SocketTools Namespace | Trace Property (SocketTools.TextMessage) | TraceFile Property (SocketTools.TextMessage) | TraceFlags Property (SocketTools.TextMessage)

TextMessage.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub TextMessage.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void TextMessage.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ErrorEventArgs that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.TextMessage (in SocketTools.TextMessage.dll)

See Also

SocketTools Namespace

TextMessage.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see TextMessage.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.TextMessage.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class TextMessage.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class TextMessage.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the TextMessage class.

Example

```
<Assembly: SocketTools.TextMessage.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.TextMessage.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.TextMessage (in SocketTools.TextMessage.dll)

See Also

TextMessage.RuntimeLicenseAttribute Members | SocketTools Namespace

TextMessage.RuntimeLicenseAttribute Members

TextMessage.RuntimeLicenseAttribute overview

Public Instance Constructors

TextMessage.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

TextMessage.RuntimeLicenseAttribute Class | SocketTools Namespace

TextMessage.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public TextMessage.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the TextMessage class.

See Also

TextMessage.RuntimeLicenseAttribute Class | SocketTools Namespace

TextMessage.RuntimeLicenseAttribute Properties

The properties of the **TextMessage.RuntimeLicenseAttribute** class are listed below. For a complete list of **TextMessage.RuntimeLicenseAttribute** class members, see the TextMessage.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

TextMessage.RuntimeLicenseAttribute Class | SocketTools Namespace

TextMessage.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

TextMessage.RuntimeLicenseAttribute Class | SocketTools Namespace

TextMessageException Class

The exception that is thrown when a client error occurs.

For a list of all members of this type, see TextMessageException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.TextMessageException

[Visual Basic]
```
Public Class TextMessageException
    Inherits ApplicationException
```

[C#]
```
public class TextMessageException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A TextMessageException is thrown by the TextMessage class when an error occurs.

The default constructor for the TextMessageException class sets the **ErrorCode** property to the last client error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.TextMessage (in SocketTools.TextMessage.dll)

See Also

TextMessageException Members | SocketTools Namespace

TextMessageException Members

TextMessageException overview

Public Instance Constructors

TextMessageException	Overloaded. Initializes a new instance of the TextMessageException class.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

TextMessageException Class | SocketTools Namespace

TextMessageException Constructor

Initializes a new instance of the TextMessageException class with the last network error code.

Overload List

Initializes a new instance of the TextMessageException class with the last network error code.

public TextMessageException();

Initializes a new instance of the TextMessageException class with a specified error number.

public TextMessageException(int);

Initializes a new instance of the TextMessageException class with a specified error message.

public TextMessageException(string);

Initializes a new instance of the TextMessageException class with a specified error message and a reference to the inner exception that is the cause of this exception.

public TextMessageException(string,Exception);

See Also

TextMessageException Class | SocketTools Namespace

TextMessageException Constructor ()

Initializes a new instance of the TextMessageException class with the last network error code.

[Visual Basic]

```
Overloads Public Sub New()
```

[C#]

```
public TextMessageException();
```

Remarks

The ctor constructor sets the **ErrorCode** property to the last client error that occurred. For more information about the errors that may occur, refer to the TextMessage.ErrorCode enumeration.

See Also

TextMessageException Class | SocketTools Namespace | TextMessageException Constructor Overload List

TextMessageException Constructor (Int32)

Initializes a new instance of the TextMessageException class with a specified error number.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

[C#]

```
public TextMessageException(
   int code
);
```

Parameters

code

An integer value which specifies an error code.

Remarks

This constructor sets the **ErrorCode** property to the specified error code. For more information about the errors that may occur, refer to the TextMessage.ErrorCode enumeration.

See Also

TextMessageException Class | SocketTools Namespace | TextMessageException Constructor Overload List

TextMessageException Constructor (String)

Initializes a new instance of the TextMessageException class with a specified error message.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]

```
public TextMessageException(
   string message
);
```

Parameters

message

The error message that explains the reason for the exception.

Remarks

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

TextMessageException Class | SocketTools Namespace | TextMessageException Constructor Overload List

TextMessageException Constructor (String, Exception)

Initializes a new instance of the TextMessageException class with a specified error message and a reference to the inner exception that is the cause of this exception.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal innerException As Exception _
)
```

[C#]

```
public TextMessageException(
   string message,
   Exception innerException
);
```

Parameters

message
The error message that explains the reason for the exception.

innerException
The exception that is the cause of the current exception. If the *innerException* parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

TextMessageException Class | SocketTools Namespace | TextMessageException Constructor Overload List

TextMessageException Properties

The properties of the **TextMessageException** class are listed below. For a complete list of **TextMessageException** class members, see the TextMessageException Members topic.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

TextMessageException Class | SocketTools Namespace

TextMessageException.ErrorCode Property

Gets a value which specifies the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property ErrorCode As ErrorCode
```

[C#]

```
public TextMessage.ErrorCode ErrorCode {get;}
```

Property Value

Returns a TextMessage.ErrorCode enumeration value which specifies the error code.

Remarks

The **ErrorCode** property returns the error code that specifies the cause of the exception.

The default constructor for the TextMessageException class sets the error code to the last network error that occurred. For more information about the errors that may occur, refer to the TextMessage.ErrorCode enumeration.

See Also

TextMessageException Class | SocketTools Namespace

TextMessageException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]

```
Overrides Public ReadOnly Property Message As String
```

[C#]

```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

TextMessageException Class | SocketTools Namespace

TextMessageException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception. For network related errors, this value is the same as the values returned by the Windows Sockets API. For more information about socket error codes, see the Windows Socket Version 2 API error code documentation in MSDN.

See Also

TextMessageException Class | SocketTools Namespace

TextMessageException Methods

The methods of the **TextMessageException** class are listed below. For a complete list of **TextMessageException** class members, see the TextMessageException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

TextMessageException Class | SocketTools Namespace

TextMessageException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]
```
Overrides Public Function ToString() As String
```

[C#]
```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

TextMessageException Class | SocketTools Namespace

WebLocation Class

Get physical location information for the local computer system.

For a list of all members of this type, see WebLocation Members.

System.Object
  SocketTools.WebLocation

[Visual Basic]

```
Public Class WebLocation
    Implements IDisposable
```

[C#]

```
public class WebLocation : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The **SocketTools.WebLocation** class returns information about the location associated with the with the external IP address of the local system. The accuracy of this information can vary depending on the location, with the most detailed information being available for North America. The country and time zone information for all locations is generally accurate. However, as the location information becomes more precise, details such as city names, postal codes and specific geographic locations (e.g.: longitude and latitude) may have reduced accuracy.

This location information should not be used by applications which require extremely accurate map coordinates, such as navigation software. The location information in North America should be generally accurate within a 25 mile (40km) radius. However, given the nature of how IP address location works, there is no guarantee that location information for any specific IP address or network will be accurate.

Software that is designed to protect the privacy of users, such as those which route all Internet traffic through proxy servers or VPNs, can significantly impact the accuracy of this information. In this case, the data returned by this class may reflect the location of the network or proxy server, and not the location of the person using your application. It is recommended that you always request permission from the user before acquiring their location, have them confirm that the location is correct and provide a mechanism for them to update that information.

This component uses SocketTools Web Services and will only function if there is an active Internet connection and the local system is capable of establishing a secure connection to the location service.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WebLocation (in SocketTools.WebLocation.dll)

See Also

WebLocation Members | SocketTools Namespace

WebLocation Members

WebLocation overview

Public Instance Constructors

WebLocation Constructor	Initializes a new instance of the WebLocation class.

Public Instance Properties

ASNumber	Gets the autonomous system number for the network.
CityName	Gets the city name for the current location.
Coordinates	Gets the GPS coordinates for the current location.
CountryAlpha	Gets the ISO code for the country at the current location.
CountryCode	Gets the numeric code for the country at the current location.
CountryName	Gets the name of the country for the current location.
IPAddress	Gets the external IP address for the local system.
IsInitialized	Determine if the class instance has been initialized.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LastUpdate	Gets the number of seconds since the last location update.
Latitude	Gets the latitude for the current location.
LocalTime	Gets the local time adjusted for the location's time zone.
LocationId	Gets a unique identifier for the current location.
Longitude	Gets the longitude for the current location
Organization	Gets the name of the organization that owns the network.
PostalCode	Gets the postal code for the current location.
RegionCode	Gets the numeric region code for the current location.
RegionName	Gets the region name for the current location
Subdivision	Gets the subdivision name for the current location.
SubdivisionCode	Gets the alphanumeric code for the current subdivision.
ThrowError	Gets and sets a value which specifies if method

	calls should throw exceptions when an error occurs.
Timeout	Gets and sets the number of seconds to wait until a location query fails.
Timezone	Gets the time zone for the current location.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
TzOffset	Gets the time zone offset in seconds for the current location.
TzShortName	Gets an abbreviated time zone name for the current location.
Version	Gets a value which returns the current version of the WebLocation class library.

Public Instance Methods

Dispose	Overloaded. Releases all resources used by WebLocation.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the WebLocation class.
Reset	Reset the internal state of the object, resetting all properties to their default values.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
Update	Update the current location information for the local system.

Public Instance Events

OnError	Occurs when an client operation fails.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the WebLocation class and optionally

	releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

WebLocation Class | SocketTools Namespace

WebLocation Constructor

Initializes a new instance of the WebLocation class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public WebLocation();
```

See Also

WebLocation Class | SocketTools Namespace

WebLocation Properties

The properties of the **WebLocation** class are listed below. For a complete list of **WebLocation** class members, see the WebLocation Members topic.

Public Instance Properties

ASNumber	Gets the autonomous system number for the network.
CityName	Gets the city name for the current location.
Coordinates	Gets the GPS coordinates for the current location.
CountryAlpha	Gets the ISO code for the country at the current location.
CountryCode	Gets the numeric code for the country at the current location.
CountryName	Gets the name of the country for the current location.
IPAddress	Gets the external IP address for the local system.
IsInitialized	Determine if the class instance has been initialized.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LastUpdate	Gets the number of seconds since the last location update.
Latitude	Gets the latitude for the current location.
LocalTime	Gets the local time adjusted for the location's time zone.
LocationId	Gets a unique identifier for the current location.
Longitude	Gets the longitude for the current location
Organization	Gets the name of the organization that owns the network.
PostalCode	Gets the postal code for the current location.
RegionCode	Gets the numeric region code for the current location.
RegionName	Gets the region name for the current location
Subdivision	Gets the subdivision name for the current location.
SubdivisionCode	Gets the alphanumeric code for the current subdivision.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

Timeout	Gets and sets the number of seconds to wait until a location query fails.
Timezone	Gets the time zone for the current location.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
TzOffset	Gets the time zone offset in seconds for the current location.
TzShortName	Gets an abbreviated time zone name for the current location.
Version	Gets a value which returns the current version of the WebLocation class library.

See Also

WebLocation Class | SocketTools Namespace

WebLocation.ASNumber Property

Gets the autonomous system number for the network.

[Visual Basic]

```
Public ReadOnly Property ASNumber As Integer
```

[C#]

```
public int ASNumber {get;}
```

Property Value

An integer value which uniquely identifies a global network.

Remarks

This property returns an integer value which is used to uniquely identify a global network (autonomous system) that is connected to the Internet. This number is assigned by regional registries and used by large networks, such as Internet Service Providers, for exchanging routing information with one another. This value can be used to determine the ownership of a particular network.

See Also

WebLocation Class | SocketTools Namespace | IPAddress Property | LocationId Property | Organization Property

WebLocation.CityName Property

Gets the city name for the current location.

[Visual Basic]

```
Public ReadOnly Property CityName As String
```

[C#]

```
public string CityName {get;}
```

Property Value

A string which identifies the city in which the external IP address is located.

Remarks

This property returns a string which identifies the city in which the external IP address is located. These names will always be in English, regardless of the current system locale. If the city name cannot be determined, this property may return an empty string.

See Also

WebLocation Class | SocketTools Namespace | CountryName Property | PostalCode Property | RegionName Property | Subdivision Property

WebLocation.Coordinates Property

Gets the GPS coordinates for the current location.

[Visual Basic]

```
Public ReadOnly Property Coordinates As String
```

[C#]

```
public string Coordinates {get;}
```

Property Value

A string which specifies the location coordinates.

Remarks

This property returns a string which specifies the location expressed using the Universal Transverse Mercator (UTM) coordinate system with the WGS-84 ellipsoid. UTM coordinates are commonly used with the Global Positioning System (GPS) and are comprised of three parts: the zone, the easting (the eastward-measured distance or x-coordinate) and the northing (the northward-measured distance or y-coordinate). An example of a string value returned by this property would be "14S 702089E 3646476N".

See Also

WebLocation Class | SocketTools Namespace | Latitude Property | Longitude Property

WebLocation.CountryAlpha Property

Gets the ISO code for the country at the current location.

[Visual Basic]

```
Public ReadOnly Property CountryAlpha As String
```

[C#]

```
public string CountryAlpha {get;}
```

Property Value

A string which contains the ISO 3166-1 alpha-2 code for the country.

Remarks

This property returns a string which contains the ISO 3166-1 alpha-2 code for the country the external IP address is located in. This is a two-letter country code established by the International Organization for Standardization (ISO).

The SubdivisionCode property can be used to determine the standard ISO standard code for a regional subdivision (such as a state, province or territory). For example, if the IP address is located in Los Angeles, California the CountryAlpha property would return "US" and the SubdivisionCode property would return "CA".

See Also

WebLocation Class | SocketTools Namespace | CountryCode Property | CountryName Property | RegionName Property | Subdivision Property | SubdivisionCode Property

WebLocation.CountryCode Property

Gets the numeric code for the country at the current location.

[Visual Basic]
```
Public ReadOnly Property CountryCode As Integer
```

[C#]
```
public int CountryCode {get;}
```

Property Value

An integer value which identifies the country where the external IP address is located.

Remarks

This property returns an integer value which identifies the country where the external IP address is located. These codes can be up to three digits (usually displayed with leading zeros as necessary) and correspond to the country codes assigned by the United Nations. For example, the code for the United States is 840. It is important to note that these are not international dialing codes and should not be used with telephony applications.

See Also

WebLocation Class | SocketTools Namespace | CountryAlpha Property | CountryName Property | RegionName Property | Subdivision Property

WebLocation.CountryName Property

Gets the name of the country for the current location.

[Visual Basic]

```
Public ReadOnly Property CountryName As String
```

[C#]

```
public string CountryName {get;}
```

Property Value

A string which contains the full name of the country.

Remarks

This property returns a string which contains the full name of the country in which the external IP address is located. These names will always be in English, regardless of the current system locale.

See Also

WebLocation Class | SocketTools Namespace | CountryAlpha Property | CountryCode Property | RegionName Property | Subdivision Property

WebLocation.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

WebLocation.ErrorEventArgs Class | SocketTools Namespace | Error Property

WebLocation.IPAddress Property

Gets the external IP address for the local system.

[Visual Basic]

```
Public ReadOnly Property IPAddress As String
```

[C#]

```
public string IPAddress {get;}
```

Property Value

a string which specifies the external IP address for the local system.

Remarks

This property returns a string which specifies the external IP address for the local system. If the system has been assigned multiple IP addresses, it reflects the address of the interface that was used to establish a connection with the SocketTools server. If the connection is made through a Virtual Private Network (VPN) it will use that assigned IP address. If a connection is made through a proxy server, the IP address may be address of the proxy rather than the local host, depending on how the connection is made.

This property cannot be assigned a value to query the location of different IP addresses. If the external IP address of the local system cannot be determined, this property will return an empty string.

See Also

WebLocation Class | SocketTools Namespace | ASNumber Property | LocationId Property | Organization Property

WebLocation.IsInitialized Property

Determine if the class instance has been initialized.

[Visual Basic]

```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]

```
public bool IsInitialized {get;}
```

Property Value

A boolean value which specifies if the class instance has been initialized.

Remarks

This property is used to determine if the current instance of the class has been initialized properly. Normally this is done by calling the **Initialize** method after the class has been instantiated. If this property returns **false**, the application must call the **Initialize** method with a valid runtime license key before performing any other operation.

The most common reason that the class instance may not initialize correctly is that no valid development or runtime license key can be found or the license key that was provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required networking libraries or not being able to access the system registry

See Also

WebLocation Class | SocketTools Namespace | Initialize Method

WebLocation.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public WebLocation.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

WebLocation Class | SocketTools Namespace

WebLocation.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

WebLocation Class | SocketTools Namespace

WebLocation.LastUpdate Property

Gets the number of seconds since the last location update.

[Visual Basic]

```
Public ReadOnly Property LastUpdate As Integer
```

[C#]

```
public int LastUpdate {get;}
```

Property Value

An integer value which specifies the number of seconds since the last location query was performed.

Remarks

This property returns an integer value which specifies the number of seconds since the last location query was performed. A return value of zero indicates that the current location has not been updated.

See Also

WebLocation Class | SocketTools Namespace | Reset Method | Update Method

WebLocation.Latitude Property

Gets the latitude for the current location.

[Visual Basic]

```
Public ReadOnly Property Latitude As Double
```

[C#]

```
public double Latitude {get;}
```

Property Value

A numeric value which specifies the latitude of the current location in decimal format

Remarks

This property returns a value which specifies the latitude of the current location in decimal format. A positive value indicates a location that is north of the equator, while a negative value is a location that is south of the equator.

See Also

WebLocation Class | SocketTools Namespace | Coordinates Property | Longitude Property

WebLocation.LocalTime Property

Gets the local time adjusted for the location's time zone.

[Visual Basic]

```
Public ReadOnly Property LocalTime As Date
```

[C#]

```
public System.DateTime LocalTime {get;}
```

Property Value

A **DateTime** value which contains the date and time at the current location.

Remarks

This property returns the current date and time at the location, adjusted for its time zone and whether or not it's in daylight savings time.

If the current location cannot be determined, this property will return **DateTime.MinValue**.

See Also

WebLocation Class | SocketTools Namespace | Timezone Property | TzOffset Property | TzShortName Property

WebLocation.LocationId Property

Gets a unique identifier for the current location.

[Visual Basic]

```
Public ReadOnly Property LocationId As String
```

[C#]

```
public string LocationId {get;}
```

Property Value

A string of hexadecimal characters which uniquely identifies the location for this computer system

Remarks

This property returns a string of hexadecimal characters which uniquely identifies the location for this computer system. This value is used internally by the location service, and may also be used by the application for its own purposes. If this value changes in subsequent queries, it indicates the external IP address for the local system has changed.

See Also

WebLocation Class | SocketTools Namespace | ASNumber Property | IPAddress Property | Organization Property

WebLocation.Longitude Property

Gets the longitude for the current location

[Visual Basic]

```
Public ReadOnly Property Longitude As Double
```

[C#]

```
public double Longitude {get;}
```

Property Value

A numeric value which specifies the longitude of the current location in decimal format

Remarks

This property returns a value which specifies the longitude of the current location in decimal format. A positive value indicates a location that is east of the prime meridian, while a negative value is a location that is west of the prime meridian.

See Also

WebLocation Class | SocketTools Namespace | Coordinates Property | Latitude Property

WebLocation.Organization Property

Gets the name of the organization that owns the network.

[Visual Basic]

```
Public ReadOnly Property Organization As String
```

[C#]

```
public string Organization {get;}
```

Property Value

A string which identifies the organization associated with the local system's external IP address.

Remarks

This property returns a string which identifies the organization associated with the local system's external IP address. For residential end-users this is typically the name of their Internet Service provider, however it may also identify a private company such as Microsoft, Google or Amazon. Because of the nature of how this information is updated, the organization names can change over time due to acquisitions or changes of ownership. If the owner of the network cannot be determined, this property may return an empty string.

See Also

WebLocation Class | SocketTools Namespace | ASNumber Property | IPAddress Property | LocationId Property

WebLocation.PostalCode Property

Gets the postal code for the current location.

[Visual Basic]
```
Public ReadOnly Property PostalCode As String
```

[C#]
```
public string PostalCode {get;}
```

Property Value

A string which specifies the postal code for the current location.

Remarks

This property returns a string which contains the postal code associated with the area where the IP address is located. In the United States, this is a 5-digit numeric code. In Canada, this will contain the forward sortation area (the first three characters of the six character postal code). Local delivery portions of a postal code (such as the ZIP+4 code in the United States, or the local delivery unit in Canada) are not included.

This information will be most accurate within the geographic region of North America. Postal codes are locale specific, and this property may return an empty string if the postal code for the location cannot be determined.

See Also

WebLocation Class | SocketTools Namespace | CityName Property | CountryName Property | RegionName Property | Subdivision Property

WebLocation.RegionCode Property

Gets the numeric region code for the current location.

[Visual Basic]

```
Public ReadOnly Property RegionCode As Integer
```

[C#]

```
public int RegionCode {get;}
```

Property Value

An integer value which identifies the geographical region in which the external IP address is located.

Remarks

This property returns an integer value which identifies the geographical region in which the external IP address is located. This value corresponds to the name returned by the **RegionName** property. The numeric values use the UN M49 standard established by the United Nations Statistics Division.

See Also

WebLocation Class | SocketTools Namespace | CityName Property | CountryName Property | RegionName Property | Subdivision Property

WebLocation.RegionName Property

Gets the region name for the current location

[Visual Basic]

```
Public ReadOnly Property RegionName As String
```

[C#]

```
public string RegionName {get;}
```

Property Value

A string which identifies the region in which the external IP address is located.

Remarks

This property returns a string which identifies the region in which the external IP address is located. This refers to a broad geographical area, such as "North America" or "Southeast Asia" and uses the conventions for supranational regions as defined by UN M49 codes. These names will always be in English, regardless of the current system locale.

See Also

WebLocation Class | SocketTools Namespace | CityName Property | CountryName Property | RegionCode Property | Subdivision Property

WebLocation.Subdivision Property

Gets the subdivision name for the current location.

[Visual Basic]

```
Public ReadOnly Property Subdivision As String
```

[C#]

```
public string Subdivision {get;}
```

Property Value

A string which identifies the geopolitical subdivision within a country where the external IP address is located.

Remarks

This property returns a string which identifies the geopolitical subdivision within a country where the external IP address is located. In the United States, this will contain the full name of the state or commonwealth. In Canada, this will contain the name of the province or territory. These names will always be in English, regardless of the current system locale.

If a subdivision name does not exist for the location, this property will return an empty string.

The **SubdivisionCode** property will return a standardized abbreviation for the area. For example, in the United States it would return the two-character code for the state.

See Also

WebLocation Class | SocketTools Namespace | CityName Property | CountryName Property | RegionName Property | SubdivisionCode Property

WebLocation.SubdivisionCode Property

Gets the alphanumeric code for the current subdivision.

[Visual Basic]

```
Public ReadOnly Property SubdivisionCode As String
```

[C#]

```
public string SubdivisionCode {get;}
```

Property Value

A string which is either a two- or three-letter code which identifies a geopolitical subdivision within the country.

Remarks

This property returns a string string which is either a two- or three-letter code which identifies a geopolitical subdivision within the country where the external IP address is located. These codes are defined by the ISO 3166-2 standard. For example, the code for the state of California in the United States is "CA".

For international specificity, these subdivision codes are often combined with the value of the ISO alpha-2 code returned by the **CountryAlpha** property. For example, to identify the state of California, you could combine with the alpha-2 code for the United States, creating "US-CA" as an identifier.

If a subdivision code does not exist for the location, this property will return an empty string.

See Also

WebLocation Class | SocketTools Namespace | CityName Property | CountryName Property | RegionName Property | Subdivision Property

WebLocation.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]

```
Public Property ThrowError As Boolean
```

[C#]

```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

WebLocation Class | SocketTools Namespace | LastError Property | LastErrorString Property

WebLocation.Timeout Property

Gets and sets the number of seconds to wait until a location query fails.

[Visual Basic]

```
Public Property Timeout As Integer
```

[C#]

```
public int Timeout {get; set;}
```

Property Value

An integer value which specifies a timeout period in seconds.

Remarks

Setting this property specifies the number of seconds until a request for location information fails and returns an error. This includes the amount of time spent attempting to connect to the location service. If a connection cannot be established within the given time period, the operation will fail.

See Also

WebLocation Class | SocketTools Namespace | Update Method

WebLocation.Timezone Property

Gets the time zone for the current location.

[Visual Basic]

```
Public ReadOnly Property Timezone As String
```

[C#]

```
public string Timezone {get;}
```

Property Value

A string which specifies the full time zone name in which the external IP address is located.

Remarks

This property returns a string which specifies the full time zone name in which the external IP address is located. These names are defined by the Internet Assigned Numbers Authority (IANA) and have values like "America/Los_Angeles" and "Europe/London". These time zones may also be defined as "Etc/GMT+10" if there is not a regional name associated with the time zone.

The **TzShortName** property will return an abbreviated time zone name, such as "PST".

See Also

WebLocation Class | SocketTools Namespace | LocalTime Property | TzOffset Property | TzShortName Property

WebLocation.Trace Property

Gets and sets a value which indicates if network function logging is enabled.

[Visual Basic]

```
Public Property Trace As Boolean
```

[C#]

```
public bool Trace {get; set;}
```

Property Value

Returns **true** if network function tracing is enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Trace** property is used to enable (or disable) the tracing of network function calls. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the SocketTools networking classes and is enabled or disabled for an entire process. This means that once trace logging is enabled for a given component, all of the function calls made by the process using any of the SocketTools classes will be logged. For example, if you have an application using both the File Transfer Protocol and Post Office Protocol classes, and you set the **Trace** property to **true**, function calls made by both classes will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all classes used by the process.

If trace logging is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the logfile is fairly large. Since logfiles can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

When redistributing your application, make sure that you include the **SocketTools10.TraceLog.dll** module with your installation. If this library is not present, then no trace output will be generated and the value of the **Trace** property will be ignored. Only those function calls made by the SocketTools networking classes will be logged. Calls made directly to the Windows Sockets API, or calls made by other classes, will not be logged.

See Also

WebLocation Class | SocketTools Namespace

WebLocation.TraceFile Property

Gets and sets a value which specifies the name of the logfile.

[Visual Basic]

```
Public Property TraceFile As String
```

[C#]

```
public string TraceFile {get; set;}
```

Property Value

A string which specifies the name of the file.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named **SocketTools.log** is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFile** property should be set to the same value for each instance. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
 MyApp INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0 MyApp WRN:
connect(46, 192.0.0.1:1234, 16) returned -1 [10035] MyApp ERR: accept(46,
NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the component is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Note that if the specified file cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

WebLocation Class | SocketTools Namespace

WebLocation.TraceFlags Property

Gets and sets a value which specifies the client function tracing flags.

[Visual Basic]

```
Public Property TraceFlags As TraceOptions
```

[C#]

```
public WebLocation.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration which specifies the amount of detail written to the trace logfile.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Because network function tracing is enabled per-process, the trace flags are shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFlags** property should be set to the same value for each instance. Changing the trace flags for any one instance of the class will affect the logging performed for all SocketTools classes used by the application.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being written and the error **errorOperationWouldBlock** occurs, a warning is generated because the application simply needs to attempt to write the data at a later time.

See Also

WebLocation Class | SocketTools Namespace

WebLocation.TzOffset Property

Gets the time zone offset in seconds for the current location.

[Visual Basic]
```
Public ReadOnly Property TzOffset As Integer
```

[C#]
```
public int TzOffset {get;}
```

Property Value

An integer which specifies the number of seconds east or west of the prime meridian (UTC).

Remarks

This property returns an integer which specifies the number of seconds east or west of the prime meridian (UTC). A positive value indicates a time zone that is east of the prime meridian and a negative value indicates a time zone that is west of the prime meridian. For example, the Pacific time zone in the western United States during daylight savings time would have a value of -25200 (7 hours).

See Also

WebLocation Class | SocketTools Namespace | LocalTime Property | Timezone Property | TzShortName Property

WebLocation.TzShortName Property

Gets an abbreviated time zone name for the current location.

[Visual Basic]

```
Public ReadOnly Property TzShortName As String
```

[C#]

```
public string TzShortName {get;}
```

Property Value

A string which specifies the abbreviated time zone code for the current location.

Remarks

This property returns a string which specifies the abbreviated time zone code in which the external IP address is located. If daylight savings time is used within the time zone, then this value can change based on whether or not daylight savings is in effect. For example, if the IP address is located within the Pacific time zone in the United States, this will return "PDT" when daylight savings is in effect and "PST" when it is not.

If the time zone code cannot be determined for this location, a value such as "UTC+9" may be returned, indicating the number of hours ahead or behind UTC.

See Also

WebLocation Class | SocketTools Namespace | LocalTime Property | Timezone Property | TzOffset Property

WebLocation.Version Property

Gets a value which returns the current version of the WebLocation class library.

[Visual Basic]

```
Public ReadOnly Property Version As String
```

[C#]

```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the WebLocation class library. This value can be used by an application for validation and debugging purposes.

See Also

WebLocation Class | SocketTools Namespace

WebLocation Methods

The methods of the **WebLocation** class are listed below. For a complete list of **WebLocation** class members, see the WebLocation Members topic.

Public Instance Methods

Dispose	Overloaded. Releases all resources used by WebLocation.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the WebLocation class.
Reset	Reset the internal state of the object, resetting all properties to their default values.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
Update	Update the current location information for the local system.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the WebLocation class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

WebLocation Class | SocketTools Namespace

WebLocation.Dispose Method

Releases all resources used by WebLocation.

Overload List

Releases all resources used by WebLocation.

public void Dispose();

Releases the unmanaged resources allocated by the WebLocation class and optionally releases the managed resources.

protected virtual void Dispose(bool);

See Also

WebLocation Class | SocketTools Namespace

WebLocation.Dispose Method ()

Releases all resources used by WebLocation.

[Visual Basic]

```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]

```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

WebLocation Class | SocketTools Namespace | WebLocation.Dispose Overload List

WebLocation.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the WebLocation class and optionally releases the managed resources.

[Visual Basic]

```
Overridable Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]

```
protected virtual void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **WebLocation** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

WebLocation Class | SocketTools Namespace | WebLocation.Dispose Overload List

WebLocation.Initialize Method

Initialize an instance of the WebLocation class.

Overload List

Initialize an instance of the WebLocation class.

public bool Initialize();

Initialize an instance of the WebLocation class.

public bool Initialize(string);

See Also

WebLocation Class | SocketTools Namespace | Uninitialize Method | RuntimeLicenseAttribute Class

WebLocation.Initialize Method ()

Initialize an instance of the WebLocation class.

[Visual Basic]

```
Overloads Public Function Initialize() As Boolean
```

[C#]

```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The **Initialize** method can be used to explicitly initialize an instance of the **WebLocation** class, loading the networking library and allocating resources for the current thread. This must be the first method that is called before you attempt invoke other methods. Failure to initialize the class may result in subsequent errors and/or cause an exception to be raised.

Each time the **Initialize** method is invoked, it increments an internal counter that keeps track of the number of times that it has been called by any thread in the current process. The **Uninitialize** method decrements this counter, and when the usage count drops to zero, the class will automatically unload the system libraries that it has dynamically loaded and will destroy the process heap that was allocated when the first instance of the class was created.

See Also

WebLocation Class | SocketTools Namespace | WebLocation.Initialize Overload List | Uninitialize Method | RuntimeLicenseAttribute Class

WebLocation.Initialize Method (String)

Initialize an instance of the WebLocation class.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The **Initialize** method can be used to explicitly initialize an instance of the **WebLocation** class, loading the networking library and allocating resources for the current thread. This must be the first method that is called before you attempt invoke other methods. Failure to initialize the class may result in subsequent errors and/or cause an exception to be raised.

Each time the **Initialize** method is invoked, it increments an internal counter that keeps track of the number of times that it has been called by any thread in the current process. The **Uninitialize** method decrements this counter, and when the usage count drops to zero, the class will automatically unload the system libraries that it has dynamically loaded and will destroy the process heap that was allocated when the first instance of the class was created.

Example

The following example shows how to use the Initialize method to initialize an instance of the class. This example assumes that the license key string has been defined in code.

```
SocketTools.WebLocation myLocation = new SocketTools.WebLocation();

if (txtMessage.Initialize(strLicenseKey) == false)
{
 MessageBox.Show(myLocation.LastErrorString, "Error",
 MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
 return;
}
```

```
Dim myLocation As New SocketTools.WebLocation

If myLocation.Initialize(strLicenseKey) = False Then
    MsgBox(myLocation.LastErrorString, vbIconExclamation)
    Exit Sub
End If
```

See Also

WebLocation Class | SocketTools Namespace | WebLocation.Initialize Overload List | Uninitialize Method | RuntimeLicenseAttribute Class

WebLocation.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. If a socket has been allocated, it will be released and any active connections will be terminated. All properties will be reset to their default values.

See Also

WebLocation Class | SocketTools Namespace

WebLocation.Uninitialize Method

Uninitialize the class library and release any resources allocated for the current thread.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method resets the internal state of the class instance and releases the system resources it has allocated. This method is not typically used because any resources that have been allocated by an instance of the class will automatically be released when it is destroyed.

Each time the **Initialize** method is invoked, it increments an internal counter that keeps track of the number of times that it has been called by any thread in the current process. The **Uninitialize** method decrements this counter, and when the usage count drops to zero, the class will automatically unload the system libraries that it has dynamically loaded and will destroy the process heap that was allocated when the first instance of the class was created.

See Also

WebLocation Class | SocketTools Namespace | Initialize Method

WebLocation.Update Method

Update the current location information for the local system.

[Visual Basic]

```
Public Function Update() As Boolean
```

[C#]

```
public bool Update();
```

Return Value

A boolean value which indicates if the location information was updated for the local system.

Remarks

This method causes the component to query the location service to obtain current information about the physical location of the computer system based on its external IP address. The location data is cached and additional queries are only performed if it detects the external IP address for the local system has changed.

A return value of **true** means the current location was updated successfully. A return value of **false** indicates that the current location could not be determined. The **LastError** property can be used to identify the specific cause of the failure.

See Also

WebLocation Class | SocketTools Namespace | LastError Property | LastUpdate Property

WebLocation Events

The events of the **WebLocation** class are listed below. For a complete list of **WebLocation** class members, see the WebLocation Members topic.

Public Instance Events

⚡ OnError	Occurs when an client operation fails.

See Also

WebLocation Class | SocketTools Namespace

WebLocation.OnError Event

Occurs when an client operation fails.

[Visual Basic]

```
Public Event OnError As OnErrorEventHandler
```

[C#]

```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type WebLocation.ErrorEventArgs containing data related to this event. The following **WebLocation.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a client operation fails.

See Also

WebLocation Class | SocketTools Namespace | ErrorCode Enumeration | LastError Property | LastErrorString Property

WebLocation.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see WebLocation.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.WebLocation.ErrorEventArgs

[Visual Basic]
```
Public Class WebLocation.ErrorEventArgs
    Inherits EventArgs
```

[C#]
```
public class WebLocation.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WebLocation (in SocketTools.WebLocation.dll)

See Also

WebLocation.ErrorEventArgs Members | SocketTools Namespace

WebLocation.ErrorEventArgs Members

WebLocation.ErrorEventArgs overview

Public Instance Constructors

WebLocation.ErrorEventArgs Constructor	Initializes a new instance of the WebLocation.ErrorEventArgs class.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

WebLocation.ErrorEventArgs Class | SocketTools Namespace

WebLocation.ErrorEventArgs Constructor

Initializes a new instance of the WebLocation.ErrorEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public WebLocation.ErrorEventArgs();
```

See Also

WebLocation.ErrorEventArgs Class | SocketTools Namespace

WebLocation.ErrorEventArgs Properties

The properties of the **WebLocation.ErrorEventArgs** class are listed below. For a complete list of **WebLocation.ErrorEventArgs** class members, see the WebLocation.ErrorEventArgs Members topic.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

WebLocation.ErrorEventArgs Class | SocketTools Namespace

WebLocation.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public WebLocation.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

WebLocation.ErrorEventArgs Class | SocketTools Namespace | Description Property

WebLocation.ErrorCode Enumeration

Specifies the error codes returned by the WebLocation class.

[Visual Basic]
```
Public Enum WebLocation.ErrorCode
```

[C#]
```
public enum WebLocation.ErrorCode
```

Remarks

The WebLocation class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.
errorProxyAuthenticationRequired	Proxy authentication required.

errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.
errorUnsupportedMediaType	Request specified an unsupported media type.

errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.
errorNoMoreMailboxes	No more mailboxes exist on this server.

errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.
errorInvalidThreadHandle	The specified thread handle is no longer valid.

errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been defined.

errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or unrecognized.

errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WebLocation (in SocketTools.WebLocation.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

WebLocation.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub WebLocation.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void WebLocation.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ErrorEventArgs that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WebLocation (in SocketTools.WebLocation.dll)

See Also

SocketTools Namespace

WebLocation.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see WebLocation.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.WebLocation.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class WebLocation.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class WebLocation.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the WebLocation class.

Example

```
<Assembly: SocketTools.WebLocation.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.WebLocation.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.WebLocation (in SocketTools.WebLocation.dll)

See Also

WebLocation.RuntimeLicenseAttribute Members | SocketTools Namespace

WebLocation.RuntimeLicenseAttribute Members

WebLocation.RuntimeLicenseAttribute overview

Public Instance Constructors

WebLocation.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

WebLocation.RuntimeLicenseAttribute Class | SocketTools Namespace

WebLocation.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public WebLocation.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the WebLocation class.

See Also

WebLocation.RuntimeLicenseAttribute Class | SocketTools Namespace

WebLocation.RuntimeLicenseAttribute Properties

The properties of the **WebLocation.RuntimeLicenseAttribute** class are listed below. For a complete list of **WebLocation.RuntimeLicenseAttribute** class members, see the WebLocation.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

WebLocation.RuntimeLicenseAttribute Class | SocketTools Namespace

WebLocation.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

WebLocation.RuntimeLicenseAttribute Class | SocketTools Namespace

WebLocationException Class

The exception that is thrown when a client error occurs.

For a list of all members of this type, see WebLocationException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.WebLocationException

[Visual Basic]

```
Public Class WebLocationException
    Inherits ApplicationException
```

[C#]

```
public class WebLocationException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A WebLocationException is thrown by the WebLocation class when an error occurs.

The default constructor for the WebLocationException class sets the **ErrorCode** property to the last client error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WebLocation (in SocketTools.WebLocation.dll)

See Also

WebLocationException Members | SocketTools Namespace

WebLocationException Members

WebLocationException overview

Public Instance Constructors

WebLocationException	Overloaded. Initializes a new instance of the WebLocationException class.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

WebLocationException Class | SocketTools Namespace

WebLocationException Constructor

Initializes a new instance of the WebLocationException class with the last network error code.

Overload List

Initializes a new instance of the WebLocationException class with the last network error code.

public WebLocationException();

Initializes a new instance of the WebLocationException class with a specified error number.

public WebLocationException(int);

Initializes a new instance of the WebLocationException class with a specified error message.

public WebLocationException(string);

Initializes a new instance of the WebLocationException class with a specified error message and a reference to the inner exception that is the cause of this exception.

public WebLocationException(string,Exception);

See Also

WebLocationException Class | SocketTools Namespace

WebLocationException Constructor ()

Initializes a new instance of the WebLocationException class with the last network error code.

[Visual Basic]

```
Overloads Public Sub New()
```

[C#]

```
public WebLocationException();
```

Remarks

The ctor constructor sets the **ErrorCode** property to the last client error that occurred. For more information about the errors that may occur, refer to the WebLocation.ErrorCode enumeration.

See Also

WebLocationException Class | SocketTools Namespace | WebLocationException Constructor Overload List

WebLocationException Constructor (String)

Initializes a new instance of the WebLocationException class with a specified error message.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]

```
public WebLocationException(
   string message
);
```

Parameters

message

The error message that explains the reason for the exception.

Remarks

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

WebLocationException Class | SocketTools Namespace | WebLocationException Constructor Overload List

WebLocationException Constructor (String, Exception)

Initializes a new instance of the WebLocationException class with a specified error message and a reference to the inner exception that is the cause of this exception.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal innerException As Exception _
)
```

[C#]

```
public WebLocationException(
   string message,
   Exception innerException
);
```

Parameters

message

The error message that explains the reason for the exception.

innerException

The exception that is the cause of the current exception. If the ***innerException*** parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

WebLocationException Class | SocketTools Namespace | WebLocationException Constructor Overload List

WebLocationException Constructor (Int32)

Initializes a new instance of the WebLocationException class with a specified error number.

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

```
[C#]
public WebLocationException(
   int code
);
```

Parameters

code

An integer value which specifies an error code.

Remarks

This constructor sets the **ErrorCode** property to the specified error code. For more information about the errors that may occur, refer to the WebLocation.ErrorCode enumeration.

See Also

WebLocationException Class | SocketTools Namespace | WebLocationException Constructor Overload List

WebLocationException Properties

The properties of the **WebLocationException** class are listed below. For a complete list of **WebLocationException** class members, see the WebLocationException Members topic.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

WebLocationException Class | SocketTools Namespace

WebLocationException.ErrorCode Property

Gets a value which specifies the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property ErrorCode As ErrorCode
```

[C#]

```
public WebLocation.ErrorCode ErrorCode {get;}
```

Property Value

Returns a WebLocation.ErrorCode enumeration value which specifies the error code.

Remarks

The **ErrorCode** property returns the error code that specifies the cause of the exception.

The default constructor for the WebLocationException class sets the error code to the last network error that occurred. For more information about the errors that may occur, refer to the WebLocation.ErrorCode enumeration.

See Also

WebLocationException Class | SocketTools Namespace

WebLocationException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]

```
Overrides Public ReadOnly Property Message As String
```

[C#]

```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

WebLocationException Class | SocketTools Namespace

WebLocationException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception. For network related errors, this value is the same as the values returned by the Windows Sockets API. For more information about socket error codes, see the Windows Socket Version 2 API error code documentation in MSDN.

See Also

WebLocationException Class | SocketTools Namespace

WebLocationException Methods

The methods of the **WebLocationException** class are listed below. For a complete list of **WebLocationException** class members, see the WebLocationException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

WebLocationException Class | SocketTools Namespace

WebLocationException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]

```
Overrides Public Function ToString() As String
```

[C#]

```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

WebLocationException Class | SocketTools Namespace

WebStorage Class

Application storage and data management services.

For a list of all members of this type, see WebStorage Members.

System.Object
  SocketTools.WebStorage

[Visual Basic]
```
Public Class WebStorage
    Implements IDisposable
```

[C#]
```
public class WebStorage : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Example

The WebStorage class enables an application to securely store and manage data remotely.

This class uses SocketTools Web Services. It will only function if there is an active Internet connection and the local system is capable of establishing a secure connection to the storage servers.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WebStorage (in SocketTools.WebStorage.dll)

See Also

WebStorage Members | SocketTools Namespace

WebStorage Members

WebStorage overview

Public Static (Shared) Fields

webTimeout	A constant value which specifies the default timeout period.

Public Static (Shared) Methods

ErrorText	Returns the description of an error code.

Public Instance Constructors

WebStorage Constructor	Initializes a new instance of the WebStorage class.

Public Instance Properties

AccountId	Gets a value that specifies the storage account ID associated with the development license.
AppId	Gets and sets the application identifier associated with the storage container.
Handle	Gets a value that specifies the storage handle allocated for the current session.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking storage operation.
IsConnected	Gets a value which indicates if a connection to the storage server has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsOpened	Gets a value which indicates if a storage container has been opened.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
ObjectAttributes	Gets the attributes for the current object.
ObjectContent	Gets the content type for the current object.
ObjectCreated	Gets the date and time the current object was created.
ObjectDigest	Gets the SHA-256 digest value for the current object.
ObjectId	Gets the unique identifier for the current object.
ObjectLabel	Gets the label for the current object.
ObjectLimit	Gets the maximum number of objects that may be

	created.
ObjectModified	Gets the date and time the current object was last modified.
ObjectSize	Gets the size of the current object in bytes
StorageFree	Gets the total number of bytes available to store new objects.
StorageId	Gets a unique identifier for the current storage container.
StorageLimit	Gets the maximum number of bytes of storage available.
StorageObjects	Gets the current number of objects stored for your account.
StorageType	Gets the current storage container type identifier.
StorageUsed	Gets the current number of bytes of storage used for your account.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
TransferBytes	Gets a value which specifies the number of bytes transferred to or from the server.
TransferRate	Gets a value which specifies the data transfer rate in bytes per second.
TransferTime	Gets a value which specifies the number of seconds elapsed during a data transfer.
Version	Gets a value which returns the current version of the WebStorage class library.

Public Instance Methods

Cancel	Cancel the current storage operation.
Close	Close the storage container.
CompareData	Overloaded. Compare the data in a byte buffer with a stored object.
CompareFile	Compare the data in a file with a stored object.
Copy	Overloaded. Copy the contents of a stored object

	to another container.
Delete	Delete a stored object from the container.
DeleteAll	Delete all stored objects in the container.
Dispose	Overloaded. Releases all resources used by WebStorage.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
Exists	Determine if a stored object exists in the container.
FindFirst	Overloaded. Find the first stored object that matches a label or content type.
FindNext	Find the next stored object that matches a label or content type.
GetData	Overloaded. Download the data in a stored object to a byte array buffer
GetFile	Download the data in a stored object to a local file.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the WebStorage class.
Move	Move the contents of a stored object to another container.
Open	Overloaded. Open a storage container for the specified application.
PutData	Overloaded. Upload the data in a byte array buffer to the storage container.
PutFile	Overloaded. Upload a local file to the current storage container.
RegisterId	Register a new application identifier with the storage service.
Rename	Change the label of an existing storage object.
Reset	Reset the internal state of the class instance, resetting all properties to their default values.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
UnregisterId	Unregister a previously registered application identifier.
ValidateId	Check an application identifier to ensure it is valid

	and exists.
ValidateLabel	Validate an object label to ensure it uses allowed characters.

Public Instance Events

OnCancel	Occurs when a storage operation is canceled.
OnDownload	Occurs when a storage object has been downloaded.
OnError	Occurs when an storage operation fails.
OnProgress	Occurs as a data stream is being read or written to the client.
OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
OnUpload	Occurs when a storage object has been uploaded.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the WebStorage class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

WebStorage Class | SocketTools Namespace

WebStorage Constructor

Initializes a new instance of the WebStorage class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public WebStorage();
```

See Also

WebStorage Class | SocketTools Namespace

WebStorage Properties

The properties of the **WebStorage** class are listed below. For a complete list of **WebStorage** class members, see the WebStorage Members topic.

Public Instance Properties

AccountId	Gets a value that specifies the storage account ID associated with the development license.
AppId	Gets and sets the application identifier associated with the storage container.
Handle	Gets a value that specifies the storage handle allocated for the current session.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking storage operation.
IsConnected	Gets a value which indicates if a connection to the storage server has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsOpened	Gets a value which indicates if a storage container has been opened.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
ObjectAttributes	Gets the attributes for the current object.
ObjectContent	Gets the content type for the current object.
ObjectCreated	Gets the date and time the current object was created.
ObjectDigest	Gets the SHA-256 digest value for the current object.
ObjectId	Gets the unique identifier for the current object.
ObjectLabel	Gets the label for the current object.
ObjectLimit	Gets the maximum number of objects that may be created.
ObjectModified	Gets the date and time the current object was last modified.
ObjectSize	Gets the size of the current object in bytes
StorageFree	Gets the total number of bytes available to store new objects.
StorageId	Gets a unique identifier for the current storage container.

StorageLimit	Gets the maximum number of bytes of storage available.
StorageObjects	Gets the current number of objects stored for your account.
StorageType	Gets the current storage container type identifier.
StorageUsed	Gets the current number of bytes of storage used for your account.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
TransferBytes	Gets a value which specifies the number of bytes transferred to or from the server.
TransferRate	Gets a value which specifies the data transfer rate in bytes per second.
TransferTime	Gets a value which specifies the number of seconds elapsed during a data transfer.
Version	Gets a value which returns the current version of the WebStorage class library.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.AccountId Property

Gets a value that specifies the storage account ID associated with the development license.

[Visual Basic]

```
Public ReadOnly Property AccountId As String
```

[C#]

```
public string AccountId {get;}
```

Property Value

A string which contains the account ID.

Remarks

The account ID is a string that uniquely identifies the web services account that is associated with the session. The account ID corresponds with your product serial number and runtime license key, but it is not identical to either of those values.

If you are using an evaluation license, the account ID is temporary and only valid during the evaluation period. After the evaluation period has expired, the account ID is revoked and objects stored using this ID will be deleted. It is not recommended that you store critical application data using an evaluation license.

See Also

WebStorage Class | SocketTools Namespace | AppId Property | StorageId Property

WebStorage.AppId Property

Gets and sets the application identifier associated with the storage container.

[Visual Basic]
```
Public Property AppId As String
```

[C#]
```
public string AppId {get; set;}
```

Property Value

A string which contains the application identifier.

Remarks

This property returns the current application ID. The application ID is a string that uniquely identifies the application and can only contain letters, numbers, the period and the underscore character. The default value for this property is SocketTools.Storage.Default.

You can register a unique identifier for your application using the **RegisterId** method

See Also

WebStorage Class | SocketTools Namespace | AccountId Property | StorageId Property

WebStorage.Handle Property

Gets a value that specifies the storage handle allocated for the current session.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer which represents a client handle. If there is no active connection, a value of -1 is returned.

Remarks

The **Handle** property specifies the numeric descriptor of the current storage session.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.IsBlocked Property

Gets a value which indicates if the current thread is performing a blocking storage operation.

[Visual Basic]

```
Public ReadOnly Property IsBlocked As Boolean
```

[C#]

```
public bool IsBlocked {get;}
```

Property Value

Returns **true** if the current thread is blocking, otherwise returns **false**.

Remarks

The **IsBlocked** property returns **true** if the current thread is blocked performing a storage operation. Because the Windows Sockets API only permits one blocking operation per thread of execution, this property should be checked before performing a storage operation in response to an event.

If the **IsBlocked** property returns **false**, this means there are no blocking operations on the current thread at that time. However, this does not guarantee that the next storage request will not fail. An application should always check the return value from a method and check the value of the **LastError** property if an error occurs.

Note that this property will return **true** if there is any blocking operation being performed by the current thread, regardless of whether this specific instance of the class is responsible for the blocking operation or not.

See Also

WebStorage Class | SocketTools Namespace | IsConnected Property

WebStorage.IsConnected Property

Gets a value which indicates if a connection to the storage server has been established.

[Visual Basic]

```
Public ReadOnly Property IsConnected As Boolean
```

[C#]

```
public bool IsConnected {get;}
```

Property Value

Returns **true** if the connection has been established; otherwise returns **false**.

Remarks

This property will return **false** if the **Open** method has not been called to open a storage container, or if the connection to the server has been terminated. A value of **true** indicates that the storage container has been opened and there is a valid connection to the server.

The client does not maintain a continuous, persistent connection with the storage server. The connection may be closed and reopened internally as needed. If the client session has been idle for a period of time, this property can return **false**. If the **LastError** property returns **errorNotConnected** it means the client session is valid, however it not currently connected to the storage server. The next call to store or retrieve an object will the cause the client to reconnect automatically.

See Also

WebStorage Class | SocketTools Namespace | IsBlocked Property | LastError Property | Close Method | Open Method

WebStorage.IsInitialized Property

Gets a value which indicates if the current instance of the class has been initialized successfully.

[Visual Basic]

```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]

```
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the class instance has been initialized; otherwise returns **false**.

Remarks

The **IsInitialized** property is used to determine if the current instance of the class has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the class may not be able to initialize itself.

The most common reasons that a class instance may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required networking libraries or not being able to access the system registry.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public WebStorage.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.ObjectAttributes Property

Gets the attributes for the current object.

[Visual Basic]

```
Public ReadOnly Property ObjectAttributes As Attributes
```

[C#]

```
public WebStorage.Attributes ObjectAttributes {get;}
```

Property Value

Returns an Attributes enumeration value which specifies one or more attribute flags for the current storage object.

See Also

WebStorage Class | SocketTools Namespace | Exists Method | FindFirst Method | FindNext Method

WebStorage.ObjectContent Property

Gets the content type for the current object.

[Visual Basic]
```
Public ReadOnly Property ObjectContent As String
```

[C#]
```
public string ObjectContent {get;}
```

Property Value

A string which specifies the MIME content type for the current storage object.

Remarks

This property returns a string which specifies the MIME content type for the current storage object. The content type is typically determined by the object label and evaluating the contents of the object. It is also possible for the application to explicitly specify the content type of the object when it is created.

The object content type will always be in the format **type/subtype** where the type specifies a common media type (e.g.: text, audio, video, etc.) and subtype specifies the specific content. The most common content type for text files is **text/plain**. If the content type is unknown, the default content type is **application/octet-stream**.

Text objects may also optionally include the character encoding as part of the content type. For example, if an object contains UTF-8 encoded text, the content type may be returned as text/plain; charset=utf-8. If your application is parsing the content types, you must check if a character encoding was also included in the value. Text objects that do not specify an encoding either contain ASCII or text which uses the system code page. Unicode text will always be stored using UTF-8 encoding.

See Also

WebStorage Class | SocketTools Namespace | Exists Method | PutData Method | PutFile Method

WebStorage.ObjectCreated Property

Gets the date and time the current object was created.

[Visual Basic]

```
Public ReadOnly Property ObjectCreated As Date
```

[C#]

```
public System.DateTime ObjectCreated {get;}
```

Property Value

A **DateTime** value which contains the date and time when the current object was created.

Remarks

This property returns object creation date and time using Coordinated Universal Time (UTC) and is not adjusted for the local timezone.

If there is no current object, this property will return a value of **DateTime.MinValue**.

See Also

WebStorage Class | SocketTools Namespace | ObjectModified Property | Exists Method | GetData Method | GetFile Method

WebStorage.ObjectDigest Property

Gets the SHA-256 digest value for the current object.

[Visual Basic]
```
Public ReadOnly Property ObjectDigest As String
```

[C#]
```
public string ObjectDigest {get;}
```

Property Value

A string value which specifies the digest of the object contents, computed using an SHA-256 hash.

Remarks

This property returns a string value which specifies the digest of the object contents, computed using an SHA-256 hash. The digest value is always represented as a string of hexadecimal numbers that is exactly 64 characters long. It is important to note that even a zero-length object will have a digest, which is the standard SHA-256 NULL hash value.

See Also

WebStorage Class | SocketTools Namespace | ObjectContent Property | ObjectId Property | ObjectSize Property | Exists Method

WebStorage.ObjectId Property

Gets the unique identifier for the current object.

[Visual Basic]

```
Public ReadOnly Property ObjectId As String
```

[C#]

```
public string ObjectId {get;}
```

Property Value

A string value which uniquely identifies the current object.

Remarks

This property returns a unique identifier associated with the current object. Object IDs are guaranteed to be unique for each storage object that is created by the application.

See Also

WebStorage Class | SocketTools Namespace | ObjectContent Property | ObjectLabel Property | ObjectSize Property | Exists Method

WebStorage.ObjectLabel Property

Gets the label for the current object.

[Visual Basic]
```
Public Property ObjectLabel As String
```

[C#]
```
public string ObjectLabel {get; set;}
```

Property Value

A string value which contains the label assigned to the object.

Remarks

This property returns a string which specifies the label assigned to the current object by the application. Object labels are case-sensitive and must be unique for each object. An application uses labels to reference an object with a human-recognizable name, rather than referencing them by their object ID.

Object labels are similar to Windows file names, except they are case-sensitive. The maximum length of a label string is 511 characters. Leading and trailing whitespace (spaces, tabs, linebreaks, etc.) are ignored in label names.

Illegal characters include ASCII and Unicode control characters 0 through 31, single quotes (39), double quotes (34), less than symbol (60), greater than symbol (62), pipe (124), asterisk (42) and question mark (63). It is not possible to embed null characters in the label name.

Labels can contain Unicode characters which are internally encoded as UTF-8.

See Also

WebStorage Class | SocketTools Namespace | ObjectContent Property | ObjectDigest Property | ObjectId Property | ObjectSize Property | Exists Method

WebStorage.ObjectLimit Property

Gets the maximum number of objects that may be created.

[Visual Basic]

```
Public ReadOnly Property ObjectLimit As Long
```

[C#]

```
public long ObjectLimit {get;}
```

Property Value

An integer value which specifies the maximum number of storage objects that may be created.

Remarks

This property returns an integer value which specifies the maximum number of storage objects that may be created. In addition to the limit on the total amount of storage that may be used, there is a limit on the total number of objects that may be created by all applications.

See Also

WebStorage Class | SocketTools Namespace | StorageFree Property | StorageLimit Property | StorageObjects Property | StorageUsed Property

WebStorage.ObjectModified Property

Gets the date and time the current object was last modified.

[Visual Basic]

```
Public ReadOnly Property ObjectModified As Date
```

[C#]

```
public System.DateTime ObjectModified {get;}
```

Property Value

A **DateTime** value which contains the date and time when the current object was last modified.

Remarks

This property returns object creation date and time using Coordinated Universal Time (UTC) and is not adjusted for the local timezone.

If there is no current object, this property will return a value of **DateTime.MinValue**.

See Also

WebStorage Class | SocketTools Namespace | ObjectCreated Property | Exists Method | GetData Method | GetFile Method

WebStorage.ObjectSize Property

Gets the size of the current object in bytes

[Visual Basic]

```
Public ReadOnly Property ObjectSize As Long
```

[C#]

```
public long ObjectSize {get;}
```

Property Value

An integer value that specifies the size of the current storage object in bytes.

Remarks

This property returns a value that specifies the size of the current storage object in bytes. If there is no current object, this property will return a value of zero.

See Also

WebStorage Class | SocketTools Namespace | StorageFree Property | StorageLimit Property | StorageObjects Property | StorageUsed Property

WebStorage.StorageFree Property

Gets the total number of bytes available to store new objects.

[Visual Basic]
```
Public ReadOnly Property StorageFree As Long
```

[C#]
```
public long StorageFree {get;}
```

Property Value

An integer value which specifies the number of bytes available for the storage of new objects.

Remarks

This property returns a value which specifies the number of bytes available for the storage of new objects. This value reflects the total amount of available storage across all applications registered with the development account. If this value is zero, your storage account has reached its storage limit.

If your storage quota has been exceeded, either because the total number of objects or the total bytes of storage has reached their limit, your applications will be unable to create new objects. Your application can continue to access existing objects, regardless of your current quota limits.

To free storage space, use the **Delete** method to delete individual storage objects that are no longer needed by your application, or use the **DeleteAll** method to delete all objects in the current container.

See Also

WebStorage Class | SocketTools Namespace | StorageLimit Property | StorageUsed Property | Delete Method | DeleteAll Method

WebStorage.StorageId Property

Gets a unique identifier for the current storage container.

[Visual Basic]

```
Public ReadOnly Property StorageId As String
```

[C#]

```
public string StorageId {get;}
```

Property Value

A string value which identifies the current storage container.

Remarks

This property returns a string which identifies the current storage container opened with the **Open** method. The storage ID is associated with your development license and is guaranteed to be a unique value. If no storage container has been opened, this property will return an empty string.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.StorageLimit Property

Gets the maximum number of bytes of storage available.

[Visual Basic]

```
Public ReadOnly Property StorageLimit As Long
```

[C#]

```
public long StorageLimit {get;}
```

Property Value

An integer value which specifies the maximum number of bytes of data storage available.

Remarks

This property returns a value which specifies the maximum number of bytes of data storage available. This limit applies to all applications registered with the development account. In addition to limits on the total number of bytes that can be stored, there are also limits on the total number objects which may be created, and the individual size of each object.

Storage quota limits are assigned for each SocketTools development account. Accounts that are created with an evaluation license have much lower quota limits than a standard account and should be used for testing purposes only. After the evaluation period has ended, all objects stored using the evaluation license will be deleted.

This value does not represent limits on the storage used by a specific application. Quotas limits apply to all applications that are registered with the development account, which is identified with the runtime license key passed to the **Initialize** method.

If your storage quota has been exceeded, either because the total number of objects or the total bytes of storage has reached their limit, your applications will be unable to create new objects. Your application can continue to access existing objects, regardless of your current quota limits.

To free storage space, use the **Delete** method to delete individual storage objects that are no longer needed by your application, or use the **DeleteAll** method to delete all objects in the current container.

See Also

WebStorage Class | SocketTools Namespace | ObjectLimit Property | StorageFree Property | StorageObjects Property | StorageUsed Property | Delete Method | DeleteAll Method

WebStorage.StorageObjects Property

Gets the current number of objects stored for your account.

[Visual Basic]

```
Public ReadOnly Property StorageObjects As Long
```

[C#]

```
public long StorageObjects {get;}
```

Property Value

An integer value which specifies the number of storage objects allocated for the account.

Remarks

This property returns an integer value which specifies the number of storage objects allocated for the account. This value may not exceed the total number of objects specified by the **ObjectLimit** property.

See Also

WebStorage Class | SocketTools Namespace | ObjectLimit Property | StorageFree Property | StorageUsed Property

WebStorage.StorageType Property

Gets the current storage container type identifier.

[Visual Basic]

```
Public ReadOnly Property StorageType As Container
```

[C#]

```
public WebStorage.Container StorageType {get;}
```

Property Value

A Container enumeration which identifies the current storage container type

Remarks

The storage type specifies the type of container that objects will be stored in. You can think of the storage containers as special folders which store individual objects. In most cases, we recommend using **Container.storageGlobal** which means that stored objects will be accessible to all users of your application. However, you can limit access to the stored objects based on the local domain, local machine ID or the current user SID.

If you specify anything other than global storage, objects can be orphaned if the system configuration changes. For example, if **Container.storageMachine** is specified, the objects that are stored there can only be accessed from that computer system. If the system is reconfigured (for example, the boot volume formatted and Windows is reinstalled) the unique identifier for that system will change and the previous objects that were stored by your application can no longer be accessed.

It is advisable is to store critical application data and configuration information using **Container.storageGlobal** and use other non-global storage containers for configuration information that is unique to that system and/or user which is not critical and can be easily recreated.

See Also

WebStorage Class | SocketTools Namespace | StorageId Property | Open Method

WebStorage.StorageUsed Property

Gets the current number of bytes of storage used for your account.

[Visual Basic]

```
Public ReadOnly Property StorageUsed As Long
```

[C#]

```
public long StorageUsed {get;}
```

Property Value

An integer value which specifies the total number of bytes of data allocated for all storage objects.

Remarks

This property returns a value which specifies the total number of bytes of data allocated for all storage objects. This value may not exceed the total number of bytes of storage available, which is returned by the **StorageLimit** property.

This value does not represent the storage used by a specific application. This property returns the amount of storage used by all applications that are registered with the development account, which is identified with the runtime license key passed to the **Initialize** method.

See Also

WebStorage Class | SocketTools Namespace | ObjectLimit Property | StorageFree Property | StorageLimit Property

WebStorage.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]

```
Public Property ThrowError As Boolean
```

[C#]

```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.Timeout Property

Gets and sets a value which specifies a timeout period in seconds.

[Visual Basic]

```
Public Property Timeout As Integer
```

[C#]

```
public int Timeout {get; set;}
```

Property Value

An integer value which specifies a timeout period in seconds.

Remarks

Setting the **Timeout** property specifies the number of seconds until a storage operation fails and returns an error. For most applications it is recommended the timeout period be set between 10 and 20 seconds.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.Trace Property

Gets and sets a value which indicates if network function logging is enabled.

[Visual Basic]
```
Public Property Trace As Boolean
```

[C#]
```
public bool Trace {get; set;}
```

Property Value

Returns **true** if network function tracing is enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Trace** property is used to enable (or disable) the tracing of network function calls. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the SocketTools networking classes and is enabled or disabled for an entire process. This means that once trace logging is enabled for a given component, all of the function calls made by the process using any of the SocketTools classes will be logged. For example, if you have an application using both the File Transfer Protocol and Post Office Protocol classes, and you set the **Trace** property to **true**, function calls made by both classes will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all classes used by the process.

If trace logging is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the logfile is fairly large. Since logfiles can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

When redistributing your application, make sure that you include the **SocketTools10.TraceLog.dll** module with your installation. If this library is not present, then no trace output will be generated and the value of the **Trace** property will be ignored. Only those function calls made by the SocketTools networking classes will be logged. Calls made directly to the Windows Sockets API, or calls made by other classes, will not be logged.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.TraceFile Property

Gets and sets a value which specifies the name of the logfile.

```
[Visual Basic]
Public Property TraceFile As String
```

```
[C#]
public string TraceFile {get; set;}
```

Property Value

A string which specifies the name of the file.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named **SocketTools.log** is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFile** property should be set to the same value for each instance. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
 MyApp INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0 MyApp WRN:
connect(46, 192.0.0.1:1234, 16) returned -1 [10035] MyApp ERR: accept(46,
NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the control is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Note that if the specified file cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.TraceFlags Property

Gets and sets a value which specifies the client function tracing flags.

[Visual Basic]
```
Public Property TraceFlags As TraceOptions
```

[C#]
```
public WebStorage.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration which specifies the amount of detail written to the trace logfile.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Because network function tracing is enabled per-process, the trace flags are shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFlags** property should be set to the same value for each instance. Changing the trace flags for any one instance of the class will affect the logging performed for all SocketTools classes used by the application.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being written and the error **errorOperationWouldBlock** occurs, a warning is generated because the application simply needs to attempt to write the data at a later time.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.TransferBytes Property

Gets a value which specifies the number of bytes transferred to or from the server.

[Visual Basic]

```
Public ReadOnly Property TransferBytes As Long
```

[C#]

```
public long TransferBytes {get;}
```

Property Value

An integer value which specifies the number of bytes of data transferred to or from the server.

Remarks

The **TransferBytes** property returns the number of bytes that have been copied to or from the server. If this property is read while a transfer is ongoing, the property returns the number of bytes that have been copied up to that point. If read after a transfer has completed, the total number of bytes copied is returned. This property value is reset with every data transfer.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.TransferRate Property

Gets a value which specifies the data transfer rate in bytes per second.

[Visual Basic]

```
Public ReadOnly Property TransferRate As Integer
```

[C#]

```
public int TransferRate {get;}
```

Property Value

An integer value which specifies the transfer rate in bytes per second.

Remarks

The **TransferRate** property returns the rate at which the file data is being transferred, expressed in bytes per second. If this property is read while a transfer is ongoing, it returns the current average transfer rate.

If this property is read after the transfer has completed, it returns the final transfer rate which is calculated as the total number of bytes transferred divided by the number of seconds to complete the transfer.

This property value is reset with every data transfer.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.TransferTime Property

Gets a value which specifies the number of seconds elapsed during a data transfer.

[Visual Basic]

```
Public ReadOnly Property TransferTime As Integer
```

[C#]

```
public int TransferTime {get;}
```

Property Value

An integer value which specifies the transfer time in seconds.

Remarks

The **TransferTime** property returns the number of seconds that have elapsed since the data transfer began. If the property is read while a transfer is ongoing, it returns the elapsed time. If the property is read after the transfer is complete, it returns the total number of seconds it took to transfer the data.

This property value is reset with every data transfer.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.Version Property

Gets a value which returns the current version of the WebStorage class library.

[Visual Basic]

```
Public ReadOnly Property Version As String
```

[C#]

```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the WebStorage class library. This value can be used by an application for validation and debugging purposes.

See Also

WebStorage Class | SocketTools Namespace

WebStorage Methods

The methods of the **WebStorage** class are listed below. For a complete list of **WebStorage** class members, see the WebStorage Members topic.

Public Static (Shared) Methods

ErrorText	Returns the description of an error code.

Public Instance Methods

Cancel	Cancel the current storage operation.
Close	Close the storage container.
CompareData	Overloaded. Compare the data in a byte buffer with a stored object.
CompareFile	Compare the data in a file with a stored object.
Copy	Overloaded. Copy the contents of a stored object to another container.
Delete	Delete a stored object from the container.
DeleteAll	Delete all stored objects in the container.
Dispose	Overloaded. Releases all resources used by WebStorage.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
Exists	Determine if a stored object exists in the container.
FindFirst	Overloaded. Find the first stored object that matches a label or content type.
FindNext	Find the next stored object that matches a label or content type.
GetData	Overloaded. Download the data in a stored object to a byte array buffer
GetFile	Download the data in a stored object to a local file.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the WebStorage class.
Move	Move the contents of a stored object to another container.
Open	Overloaded. Open a storage container for the specified application.

PutData	Overloaded. Upload the data in a byte array buffer to the storage container.
PutFile	Overloaded. Upload a local file to the current storage container.
RegisterId	Register a new application identifier with the storage service.
Rename	Change the label of an existing storage object.
Reset	Reset the internal state of the class instance, resetting all properties to their default values.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.
UnregisterId	Unregister a previously registered application identifier.
ValidateId	Check an application identifier to ensure it is valid and exists.
ValidateLabel	Validate an object label to ensure it uses allowed characters.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the WebStorage class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.Cancel Method

Cancel the current storage operation.

[Visual Basic]

```
Public Function Cancel() As Boolean
```

[C#]

```
public bool Cancel();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If no storage container is currently open, or the current operation cannot be canceled, the return value will be **false**.

Remarks

When the **Cancel** method is called, the current storage operation will not immediately fail. An internal flag is set which causes the blocking operation to exit with an error. This means that the application cannot cancel a data transfer and immediately issue a request for another storage object. Instead it must allow the calling stack to unwind, returning back to the blocking operation before making any further method calls.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.Close Method

Close the storage container.

[Visual Basic]

```
Public Function Close() As Boolean
```

[C#]

```
public bool Close();
```

Return Value

This method returns a Boolean value. If the storage container is closed, the return value is **true**. If no storage container is currently open, or the container cannot be closed, the return value will be **false**.

Remarks

The **Close** method should be called after all operations using the storage container have completed. The access token granted to the application will be released and the memory allocated for the session cache will be freed.

This method is automatically invoked when the object is disposed or the **Reset** method is called.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.CompareData Method

Compare the data in a byte buffer with a stored object.

Overload List

Compare the data in a byte buffer with a stored object.

public bool CompareData(string,byte[],int);

Compare the data in a MemoryStream object with a stored object.

public bool CompareData(string,MemoryStream);

Compare the text in a string with a stored object.

public bool CompareData(string,string);

See Also

WebStorage Class | SocketTools Namespace | CompareFile Method

WebStorage.CompareData Method (String, Byte[], Int32)

Compare the data in a byte buffer with a stored object.

[Visual Basic]

```
Overloads Public Function CompareData( _
   ByVal labelName As String, _
   ByVal byteBuffer As Byte(), _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool CompareData(
   string labelName,
   byte[] byteBuffer,
   int length
);
```

Parameters

labelName
A string which specifies the label of the object that should be compared against the contents of the byte array buffer.

byteBuffer
A byte array which contains the data that should be compared against the contents. of the object.

length
An integer value which specifies the number of bytes to compare with the contents of the stored object. This value must match the size of the object exactly.

Return Value

This method returns a Boolean value. If the data matches the contents of the stored object exactly, it will return **true**, otherwise it will return **false**.

Remarks

The **CompareData** method performs a binary comparison of the data in the specified byte array with the contents of the storage object on the server. The amount of data in the buffer must match the size of the stored object exactly, or this method will fail. Partial comparisons are not supported by this method.

If you need to compare the contents of a file with a stored object, use the **CompareFile** method.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.CompareData Overload List | CompareFile Method

WebStorage.CompareData Method (String, MemoryStream)

Compare the data in a MemoryStream object with a stored object.

[Visual Basic]

```
Overloads Public Function CompareData( _
   ByVal labelName As String, _
   ByVal memStream As MemoryStream _
) As Boolean
```

[C#]

```
public bool CompareData(
   string labelName,
   MemoryStream memStream
);
```

Parameters

labelName

A string which specifies the label of the object that should be compared against the contents of the MemoryStream.

memStream

A **MemoryStream** object which contains the data that should be compared against the contents of the stored object.

Return Value

This method returns a Boolean value. If the data matches the contents of the stored object exactly, it will return **true**, otherwise it will return **false**.

Remarks

The **CompareData** method performs a binary comparison of the data in the **MemoryStream** object with the contents of the storage object on the server. The amount of data in the MemoryStream must match the size of the stored object exactly, or this method will fail. Partial comparisons are not supported by this method.

If you need to compare the contents of a file with a stored object, use the **CompareFile** method.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.CompareData Overload List | CompareFile Method

WebStorage.CompareData Method (String, String)

Compare the text in a string with a stored object.

[Visual Basic]

```
Overloads Public Function CompareData( _
   ByVal labelName As String, _
   ByVal textBuffer As String _
) As Boolean
```

[C#]

```
public bool CompareData(
   string labelName,
   string textBuffer
);
```

Parameters

labelName

A string which specifies the label of the object that should be compared against the contents of the text buffer.

textBuffer

A string which contains the text that should be compared against the contents of the stored object.

Return Value

This method returns a Boolean value. If the data matches the contents of the stored object exactly, it will return **true**, otherwise it will return **false**.

Remarks

The **CompareData** method performs a comparison of the text in the string buffer with the contents of the storage object on the server. The amount of text in the string buffer must match the size of the stored object exactly, or this method will fail. Partial comparisons are not supported by this method.

All text objects are stored using UTF-8 encoding and the contents of the **textBuffer** parameter will be encoded as UTF-8 prior to being compared with the contents of the stored object. If you need to perform a binary comparison of the data, use the version of this method that accepts a byte array or **MemoryStream** object.

If you need to compare the contents of a file with a stored object, use the **CompareFile** method.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.CompareData Overload List | CompareFile Method

WebStorage.CompareFile Method

Compare the data in a file with a stored object.

[Visual Basic]

```
Public Function CompareFile( _
   ByVal labelName As String, _
   ByVal localFile As String _
) As Boolean
```

[C#]

```
public bool CompareFile(
   string labelName,
   string localFile
);
```

Parameters

labelName

A string which specifies the label of the object that should be compared against the contents of the local file.

localFile

A string which specifies the name of the file to compare against the contents of the stored object. If no path is specified in the file name, the current working directory will be used.

Return Value

This method returns a Boolean value. If the contents of the file matches the contents of the stored object exactly, it will return **true**, otherwise it will return **false**.

Remarks

The **CompareFile** method performs a binary comparison of the data in the file with the contents of the storage object on the server. The contents of the file must be identical to the contents of the stored object or the method will fail.

If you need to compare the contents of a file with a string or byte array, use the **CompareData** method.

See Also

WebStorage Class | SocketTools Namespace | CompareData Method

WebStorage.Copy Method

Copy the contents of a stored object within the current container.

Overload List

Copy the contents of a stored object within the current container.

public bool Copy(string,string);

Copy the contents of a stored object to another container.

public bool Copy(string,string,Container);

See Also

WebStorage Class | SocketTools Namespace

WebStorage.Copy Method (String, String)

Copy the contents of a stored object within the current container.

[Visual Basic]

```
Overloads Public Function Copy( _
   ByVal oldLabel As String, _
   ByVal newLabel As String _
) As Boolean
```

[C#]

```
public bool Copy(
   string oldLabel,
   string newLabel
);
```

Parameters

oldLabel
A string which specifies the label of the object that should be copied.

newLabel
A string which specifies the new label for the copied object.

Return Value

This method returns a Boolean value. If the stored object was copied, it will return **true**, otherwise it will return **false**.

Remarks

The **Copy** method is used to create a copy of an existing storage object. It may be used to duplicate an object with a different label, or it may be used to copy the object to a new storage container type.

Copied objects are assigned their own unique ID and are not linked to one another. Any subsequent changes made to the original object will not affect the copied object. Attempting to copy an object to itself or another existing object will result in an error.

This method updates the current object. Various properties such as **ObjectId** and **ObjectLabel** will reflect the values associated with the new, copied object and not the original object it was copied from.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.Copy Overload List

WebStorage.Copy Method (String, String, Container)

Copy the contents of a stored object to another container.

[Visual Basic]

```
Overloads Public Function Copy( _
   ByVal oldLabel As String, _
   ByVal newLabel As String, _
   ByVal storageType As Container _
) As Boolean
```

[C#]

```
public bool Copy(
   string oldLabel,
   string newLabel,
   Container storageType
);
```

Parameters

oldLabel

A string which specifies the label of the object that should be copied.

newLabel

A string which specifies the new label for the copied object.

storageType

A Container enumeration which specifies the storage container that the stored object should be copied to.

Return Value

This method returns a Boolean value. If the stored object was copied to the container, it will return **true**, otherwise it will return **false**.

Remarks

The **Copy** method is used to create a copy of an existing storage object. It may be used to duplicate an object with a different label, or it may be used to copy the object to a new storage container type. For example, it can copy an object originally created in the **Container.storageUser** container to a new object stored in the **Container.storageMachine** container.

Copied objects are assigned their own unique ID and are not linked to one another. Any subsequent changes made to the original object will not affect the copied object. Attempting to copy an object to itself or another existing object will result in an error.

This method updates the current object. Various properties such as **ObjectId** and **ObjectLabel** will reflect the values associated with the new, copied object and not the original object it was copied from.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.Copy Overload List

WebStorage.Delete Method

Delete a stored object from the container.

```
[Visual Basic]
Public Function Delete( _
   ByVal labelName As String _
) As Boolean
```

```
[C#]
public bool Delete(
   string labelName
);
```

Parameters

labelName
A string which specifies the label of the object that should be deleted.

Return Value

This method returns a Boolean value. If the stored object was deleted, it will return **true**, otherwise it will return **false**.

Remarks

The **Delete** method is used to delete a stored object from the container. This method permanently deletes the storage object and its associated data from the server. Deleted objects cannot be recovered by the application. To remove all objects stored in the container, use the **DeleteAll** method.

See Also

WebStorage Class | SocketTools Namespace | DeleteAll Method

WebStorage.DeleteAll Method

Delete all stored objects in the container.

[Visual Basic]

```
Public Function DeleteAll() As Boolean
```

[C#]

```
public bool DeleteAll();
```

Return Value

This method returns a Boolean value. If all stored objects in the container have been deleted, it will return **true**, otherwise it will return **false**.

Remarks

The storage container contains information for each of the objects that have been stored by the application. Each of these objects are associated with the application ID and the storage type that was specified when calling the **Open** method. The **DeleteAll** method instructs the server to remove all objects in the continer, resetting it back to its initial state.

Exercise caution when using this method. The operation is immediate and the objects that are stored in the container are permanently deleted. They cannot be recovered by your application.

See Also

WebStorage Class | SocketTools Namespace | Delete Method

WebStorage.Exists Method

Determine if a stored object exists in the container.

[Visual Basic]

```
Public Function Exists( _
   ByVal labelName As String _
) As Boolean
```

[C#]

```
public bool Exists(
   string labelName
);
```

Parameters

labelName
A string which specifies the label of the object.

Return Value

This method returns a Boolean value. If an object exists in the current container with the specified label, it will return **true**, other, otherwise it will return **false**.

Remarks

The **Exists** method is used to check for the existence of a stored object in the current container. If the object exists, various properties that return information about the current object, such as **ObjectId** and **ObjectSize** will be updated to return the metadata associated with the object.

Although storage object labels are similar to Windows file names, they are case-sensitive. When requesting information about an object, your application must specify the label name exactly as it was created. The object label cannot contain wildcard characters.

To obtain information about how much storage your applications are using and the total number of stored objects, use the **StorageUsed** and **StorageObjects** properties.

If you wish to enumerate all of the stored objects within the container, use the **FindFirst** and **FindNext** methods.

See Also

WebStorage Class | SocketTools Namespace | ObjectId Property | ObjectModified Property | StorageObjects Property | FindFirst Method | FindNext Method

WebStorage.FindFirst Method

Find the first stored object in the storage container.

Overload List

Find the first stored object in the storage container.

public bool FindFirst();

Find the first stored object that matches a label.

public bool FindFirst(string);

Find the first stored object that matches a label or content type.

public bool FindFirst(string,string);

See Also

WebStorage Class | SocketTools Namespace | Exists Method | FindNext Method

WebStorage.FindFirst Method ()

Find the first stored object in the storage container.

[Visual Basic]

```
Overloads Public Function FindFirst() As Boolean
```

[C#]

```
public bool FindFirst();
```

Return Value

This method returns a Boolean value. If a storage obgject is found in the container, this method will return **true**, otherwise it will return **false**.

Remarks

The **FindFirst** method returns information about the first object in the current storage container. It is used in conjunction with the **FindNext** method to enumerate all of the objects in the container.

If an object exists, various properties that return information about the current object, such as **ObjectId** and **ObjectSize** will be updated to return the metadata associated with the object.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.FindFirst Overload List | Exists Method | FindNext Method

WebStorage.FindFirst Method (String)

Find the first stored object that matches a label.

[Visual Basic]

```
Overloads Public Function FindFirst( _
   ByVal matchLabel As String _
) As Boolean
```

[C#]

```
public bool FindFirst(
   string matchLabel
);
```

Parameters

matchLabel

A string which specifies the value to match against the object labels in the container. The string may contain wildcard characters similar to those use with the Windows filesystem. A "?" character matches any single character, and "*" matches any number of characters in the label. If this parameter is an empty string, all objects in the container will be matched.

Return Value

This method returns a Boolean value. If a matching object label is found in the current storage container, this method will return **true**, otherwise it will return **false**.

Remarks

The **FindFirst** method returns information about the first object that matches a given label. It is used in conjunction with the **FindNext** method to enumerate all of the matching objects in the storage container.

If a matching object exists, various properties that return information about the current object, such as **ObjectId** and **ObjectSize** will be updated to return the metadata associated with the object.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.FindFirst Overload List | Exists Method | FindNext Method

WebStorage.FindFirst Method (String, String)

Find the first stored object that matches a label or content type.

[Visual Basic]

```
Overloads Public Function FindFirst( _
   ByVal matchLabel As String, _
   ByVal contentType As String _
) As Boolean
```

[C#]

```
public bool FindFirst(
   string matchLabel,
   string contentType
);
```

Parameters

matchLabel

A string which specifies the value to match against the object labels in the container. The string may contain wildcard characters similar to those use with the Windows filesystem. A "?" character matches any single character, and "*" matches any number of characters in the label. If this parameter is an empty string, all objects in the container will be matched.

contentType

A string which specifies the content type of the objects to be enumerated. If this parameter is an empty string, the content type is ignored and all matching objects are returned. If a content type is specified, it must be a valid MIME media content type designated using the standard **type/subtype** nomenclature.

Return Value

This method returns a Boolean value. If a matching object label or content type is found in the current storage container, this method will return **true**, otherwise it will return **false**.

Remarks

The **FindFirst** method returns information about the first object that matches a given label, content type or both. It is used in conjunction with the **FindNext** method to enumerate all of the matching objects in the storage container.

If a matching object exists, various properties that return information about the current object, such as **ObjectId** and **ObjectSize** will be updated to return the metadata associated with the object.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.FindFirst Overload List | Exists Method | FindNext Method

WebStorage.FindNext Method

Find the next stored object that matches a label or content type.

[Visual Basic]

```
Public Function FindNext() As Boolean
```

[C#]

```
public bool FindNext();
```

Return Value

This method returns a Boolean value. If the next matching object label or content type is found, this method will return **true**, otherwise it will return **false**.

Remarks

The **FindNext** method returns information about the next object that matches a given label, content type or both. This method may only be called after the FindFirst method is called, otherwise it will fail.

If a matching object exists, various properties that return information about the current object, such as **ObjectId** and **ObjectSize** will be updated to return the metadata associated with the object.

See Also

WebStorage Class | SocketTools Namespace | Exists Method | FindFirst Method

WebStorage.GetData Method

Download the data in a stored object to a byte array buffer

Overload List

Download the data in a stored object to a byte array buffer

public bool GetData(string,byte[],ref int);

Download the data in a stored object to a byte array buffer

public bool GetData(string,ref byte[]);

Download the data in a stored object to a MemoryStream.

public bool GetData(string,MemoryStream);

Download the data in a stored object to a MemoryStream.

public bool GetData(string,ref string);

See Also

WebStorage Class | SocketTools Namespace

WebStorage.GetData Method (String, Byte[], Int32)

Download the data in a stored object to a byte array buffer

[Visual Basic]

```
Overloads Public Function GetData( _
   ByVal labelName As String, _
   ByVal byteBuffer As Byte(), _
   ByRef length As Integer _
) As Boolean
```

[C#]

```
public bool GetData(
   string labelName,
   byte[] byteBuffer,
   ref int length
);
```

Parameters

labelName
A string which specifies the label of the object that should be retrieved from the server.

byteBuffer
A byte array that will contain the object data when the method returns.

length
An integer variable passed by reference that will contain the number of bytes of data copied into the buffer. If the variable is initialized to a non-zero value prior to calling this method, this specifies the maximum number of bytes of data that may be copied into the buffer.

Return Value

This method returns a Boolean value. If the object exists and could be retrieved from the server, this method will return **true**, otherwise it will return **false**.

Remarks

The **GetData** method transfers data from a stored object to the specified buffer. This method will cause the current thread to block until the data transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

When this method returns, various properties that contain information about the current object, such as **ObjectId** and **ObjectSize**, will be updated to return the metadata associated with the object which was downloaded.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.GetData Overload List

WebStorage.GetData Method (String, Byte[])

Download the data in a stored object to a byte array buffer

[Visual Basic]

```
Overloads Public Function GetData( _
   ByVal labelName As String, _
   ByRef byteBuffer As Byte() _
) As Boolean
```

[C#]

```
public bool GetData(
   string labelName,
   ref byte[] byteBuffer
);
```

Parameters

labelName

A string which specifies the label of the object that should be retrieved from the server.

byteBuffer

A byte array variable passed by reference that will contain the object data when the method returns.

Return Value

This method returns a Boolean value. If the object exists and could be retrieved from the server, this method will return **true**, otherwise it will return **false**.

Remarks

The **GetData** method transfers data from a stored object to the specified buffer. This method will cause the current thread to block until the data transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

When this method returns, various properties that contain information about the current object, such as **ObjectId** and **ObjectSize**, will be updated to return the metadata associated with the object which was downloaded.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.GetData Overload List

WebStorage.GetData Method (String, MemoryStream)

Download the data in a stored object to a MemoryStream.

[Visual Basic]

```
Overloads Public Function GetData( _
   ByVal labelName As String, _
   ByVal memStream As MemoryStream _
) As Boolean
```

[C#]

```
public bool GetData(
   string labelName,
   MemoryStream memStream
);
```

Parameters

labelName

A string which specifies the label of the object that should be retrieved from the server.

memStream

A System.IO.MemoryStream object that will contain the object data when the method returns. This stream must be open and writable.

Return Value

This method returns a Boolean value. If the object exists and could be retrieved from the server, this method will return **true**, otherwise it will return **false**.

Remarks

The **GetData** method transfers data from a stored object to the specified buffer. This method will cause the current thread to block until the data transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

When this method returns, various properties that contain information about the current object, such as **ObjectId** and **ObjectSize**, will be updated to return the metadata associated with the object which was downloaded.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.GetData Overload List

WebStorage.GetData Method (String, String)

Download the data in a stored object to a MemoryStream.

[Visual Basic]

```
Overloads Public Function GetData( _
   ByVal labelName As String, _
   ByRef textBuffer As String _
) As Boolean
```

[C#]

```
public bool GetData(
   string labelName,
   ref string textBuffer
);
```

Parameters

labelName

A string which specifies the label of the object that should be retrieved from the server.

textBuffer

A string variable passed by reference which will contain the the contents of the stored object when the method returns.

Return Value

This method returns a Boolean value. If the object exists and could be retrieved from the server, this method will return **true**, otherwise it will return **false**.

Remarks

The **GetData** method transfers data from a stored object to the specified string. This method will cause the current thread to block until the data transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

This version of the method should only be used with objects which only contain text. It should never be used to retrieve the contents of a stored object which contains binary data.

When this method returns, various properties that contain information about the current object, such as **ObjectId** and **ObjectSize**, will be updated to return the metadata associated with the object which was downloaded.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.GetData Overload List

WebStorage.GetFile Method

Download the data in a stored object to a local file.

[Visual Basic]

```
Public Function GetFile( _
   ByVal localFile As String, _
   ByVal labelName As String _
) As Boolean
```

[C#]

```
public bool GetFile(
   string localFile,
   string labelName
);
```

Parameters

localFile

A string which specifies the name of the local file that will be created or overwritten with the contents of the storage object. If a path is not specified, the file will be created in the current working directory.

labelName

A string which specifies specifies the label of the object that should be retrieved from the server.

Return Value

This method returns a Boolean value. If the object exists and could be retrieved from the server, this method will return **true**, otherwise it will return **false**.

Remarks

The **GetFile** method downloads data from a stored object to a local file. This method will cause the current thread to block until the data transfer completes, a timeout occurs or the transfer is canceled. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

When this method returns, various properties that contain information about the current object, such as **ObjectId** and **ObjectSize**, will be updated to return the metadata associated with the object which was downloaded.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.Initialize Method

Initialize an instance of the WebStorage class.

Overload List

Initialize an instance of the WebStorage class.

public bool Initialize();

Initialize an instance of the WebStorage class.

public bool Initialize(string);

See Also

WebStorage Class | SocketTools Namespace | Uninitialize Method

WebStorage.Initialize Method ()

Initialize an instance of the WebStorage class.

[Visual Basic]

```
Overloads Public Function Initialize() As Boolean
```

[C#]

```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The **Initialize** method can be used to explicitly initialize an instance of the WebStorage class, loading the networking library and allocating resources for the current thread. Typically it is not necessary to explicitly call this method because the instance of the class is initialized by the class constructor. However, if the **Uninitialize** method is called, the class must be re-initialized before any other methods are called.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.Initialize Overload List | Uninitialize Method

WebStorage.Initialize Method (String)

Initialize an instance of the WebStorage class.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The **Initialize** method can be used to explicitly initialize an instance of the WebStorage class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the WebStorage class.

Example

The following example shows how to use the Initialize method to initialize an instance of the class. This example assumes that the license key string has been defined in code.

```
SocketTools.WebStorage webClient = new SocketTools.WebStorage();

if (webClient.Initialize(strLicenseKey) == false)
{
    MessageBox.Show(webClient.LastErrorString, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    return;
}
```

```
Dim webClient As New SocketTools.WebStorage

If webClient.Initialize(strLicenseKey) = False Then
    MsgBox(httpClient.LastErrorString, vbIconExclamation)
    Exit Sub
End If
```

See Also

WebStorage Class | SocketTools Namespace | WebStorage.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

WebStorage.Move Method

Move the contents of a stored object to another container.

[Visual Basic]

```
Public Function Move( _
   ByVal oldLabel As String, _
   ByVal newLabel As String, _
   ByVal storageType As Container _
) As Boolean
```

[C#]

```
public bool Move(
   string oldLabel,
   string newLabel,
   Container storageType
);
```

Parameters

oldLabel

A string which specifies the label of the object that should be moved.

newLabel

A string which specifies the new label for the object being moved.

storageType

A Container enumeration which specifies the storage container that the stored object should be moved to.

Return Value

This method returns a Boolean value. If the stored object was moved to the container, it will return **true**, otherwise it will return **false**.

Remarks

The **Move** method is used to move an existing storage object to a new container. For example, it can move an object originally created in the **Container.storageUser** container to the **Container.storageMachine** container. Using this method to move an object within the same container is effectively the same as calling the **Rename** method.

This method updates the current object. Various properties such as **ObjectId** and **ObjectLabel** will reflect the values associated with the object which has been moved.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.Open Method

Open the global storage container for the current application.

Overload List

Open the global storage container for the current application.

public bool Open();

Open a storage container for the current application.

public bool Open(Container);

Open the global storage container for the specified application.

public bool Open(string);

Open a storage container for the specified application.

public bool Open(string,Container);

Open a storage container for the specified application.

public bool Open(string,Container,int);

See Also

WebStorage Class | SocketTools Namespace | Close Method | Container Enumeration

WebStorage.Open Method ()

Open the global storage container for the current application.

[Visual Basic]

```
Overloads Public Function Open() As Boolean
```

[C#]

```
public bool Open();
```

Return Value

This method returns a Boolean value. If the container was opened, it will return **true**, otherwise it will return **false**.

Remarks

The **Open** method opens the global storage container and requests an access token for the application. This method that must be called prior to accessing any stored objects.

It is advisable is to store critical application data and configuration information in the global storage container and use other non-global storage containers for configuration information that is unique to that system and/or user which is not critical and can be easily recreated.

If a storage container was opened prior to calling this method, the current container will be closed and the new container opened.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.Open Overload List | Close Method | Container Enumeration

WebStorage.Open Method (Container)

Open a storage container for the current application.

[Visual Basic]

```
Overloads Public Function Open( _
   ByVal storageType As Container _
) As Boolean
```

[C#]

```
public bool Open(
   Container storageType
);
```

Parameters

storageType

A Container enumeration which specifies the storage container.

Return Value

This method returns a Boolean value. If the container was opened, it will return **true**, otherwise it will return **false**.

Remarks

The **Open** method opens the specified storage container using the current application ID and requests an access token for the application. This method that must be called prior to accessing any stored objects.

The ***storageType*** parameter specifies the type of container that objects will be stored in. In most cases, we recommend using **Container.storageGlobal** which means that stored objects will be accessible to all users of your application. However, you can limit access to the stored objects based on the local domain, local machine ID or the current user SID.

If you specify anything other than global storage, objects can be orphaned if the system configuration changes. For example, if **Container.storageMachine** is specified, the objects that are stored there can only be accessed from that computer system. If the system is reconfigured (for example, the boot volume formatted and Windows is reinstalled) the unique identifier for that system will change and the previous objects that were stored by your application wil no longer be accessible.

It is advisable is to store critical application data and configuration information in the **Container.storageGlobal** container and use other non-global storage containers for configuration information that is unique to that system and/or user which is not critical and can be easily recreated.

If a storage container was opened prior to calling this method, the current container will be closed and the new container opened.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.Open Overload List | Close Method | Container Enumeration

WebStorage.Open Method (String)

Open the global storage container for the specified application.

[Visual Basic]

```
Overloads Public Function Open( _
   ByVal appId As String _
) As Boolean
```

[C#]

```
public bool Open(
   string appId
);
```

Parameters

appId

A string which specifies the application ID for the storage container. The application ID is a string that uniquely identifies the application and can only contain letters, numbers, the period and the underscore character. will be used.

Return Value

This method returns a Boolean value. If the container was opened, it will return **true**, otherwise it will return **false**.

Remarks

The **Open** method opens the specified storage container and requests an access token for the application. This method that must be called prior to accessing any stored objects.

The application ID is a string that uniquely identifies the application requesting the access and must have been previously registered with the server by calling the **RegisterId** method. If the ***appId*** parameter is an empty string, the a default internal ID will be used which is allocated for each storage account. You can use this default ID if you wish to share data between all of the applications you create.

The ***storageType*** parameter specifies the type of container that objects will be stored in. In most cases, we recommend using **Container.storageGlobal** which means that stored objects will be accessible to all users of your application. However, you can limit access to the stored objects based on the local domain, local machine ID or the current user SID.

If you specify anything other than global storage, objects can be orphaned if the system configuration changes. For example, if **Container.storageMachine** is specified, the objects that are stored there can only be accessed from that computer system. If the system is reconfigured (for example, the boot volume formatted and Windows is reinstalled) the unique identifier for that system will change and the previous objects that were stored by your application wil no longer be accessible.

It is advisable is to store critical application data and configuration information in the **Container.storageGlobal** container and use other non-global storage containers for configuration information that is unique to that system and/or user which is not critical and can be easily recreated.

If a storage container was opened prior to calling this method, the current container will be closed and the new container opened.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.Open Overload List | Close Method | Container Enumeration

WebStorage.Open Method (String, Container)

Open a storage container for the specified application.

[Visual Basic]

```
Overloads Public Function Open( _
   ByVal appId As String, _
   ByVal storageType As Container _
) As Boolean
```

[C#]

```
public bool Open(
   string appId,
   Container storageType
);
```

Parameters

appId

A string which specifies the application ID for the storage container. The application ID is a string that uniquely identifies the application and can only contain letters, numbers, the period and the underscore character. will be used.

storageType

A Container enumeration which specifies the storage container. The default container is **Container.storageGlobal**.

Return Value

This method returns a Boolean value. If the container was opened, it will return **true**, otherwise it will return **false**.

Remarks

The **Open** method opens the specified storage container and requests an access token for the application. This method that must be called prior to accessing any stored objects.

The application ID is a string that uniquely identifies the application requesting the access and must have been previously registered with the server by calling the **RegisterId** method. If the ***appId*** parameter is an empty string, the a default internal ID will be used which is allocated for each storage account. You can use this default ID if you wish to share data between all of the applications you create.

The ***storageType*** parameter specifies the type of container that objects will be stored in. In most cases, we recommend using **Container.storageGlobal** which means that stored objects will be accessible to all users of your application. However, you can limit access to the stored objects based on the local domain, local machine ID or the current user SID.

If you specify anything other than global storage, objects can be orphaned if the system configuration changes. For example, if **Container.storageMachine** is specified, the objects that are stored there can only be accessed from that computer system. If the system is reconfigured (for example, the boot volume formatted and Windows is reinstalled) the unique identifier for that system will change and the previous objects that were stored by your application wil no longer be accessible.

It is advisable is to store critical application data and configuration information in the **Container.storageGlobal** container and use other non-global storage containers for configuration information that is unique to that system and/or user which is not critical and can be easily recreated.

If a storage container was opened prior to calling this method, the current container will be closed and the new container opened.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.Open Overload List | Close Method | Container Enumeration

WebStorage.Open Method (String, Container, Int32)

Open a storage container for the specified application.

[Visual Basic]

```
Overloads Public Function Open( _
   ByVal appId As String, _
   ByVal storageType As Container, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Open(
   string appId,
   Container storageType,
   int timeout
);
```

Parameters

appId

A string which specifies the application ID for the storage container. The application ID is a string that uniquely identifies the application and can only contain letters, numbers, the period and the underscore character. will be used.

storageType

A Container enumeration which specifies the storage container. The default container is **Container.storageGlobal**.

timeout

Return Value

This method returns a Boolean value. If the container was opened, it will return **true**, otherwise it will return **false**.

Remarks

The **Open** method opens the specified storage container and requests an access token for the application. This method that must be called prior to accessing any stored objects.

The application ID is a string that uniquely identifies the application requesting the access and must have been previously registered with the server by calling the **RegisterId** method. If the ***appId*** parameter is an empty string, the a default internal ID will be used which is allocated for each storage account. You can use this default ID if you wish to share data between all of the applications you create.

The ***storageType*** parameter specifies the type of container that objects will be stored in. In most cases, we recommend using **Container.storageGlobal** which means that stored objects will be accessible to all users of your application. However, you can limit access to the stored objects based on the local domain, local machine ID or the current user SID.

If you specify anything other than global storage, objects can be orphaned if the system configuration changes. For example, if **Container.storageMachine** is specified, the objects that are stored there can only be accessed from that computer system. If the system is reconfigured (for example, the boot volume formatted and Windows is reinstalled) the unique identifier for that system will change and the previous objects that were stored by your application wil no longer be accessible.

It is advisable is to store critical application data and configuration information in the **Container.storageGlobal** container and use other non-global storage containers for configuration information that is unique to that system and/or user which is not critical and can be easily recreated.

If a storage container was opened prior to calling this method, the current container will be closed and the new container opened.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.Open Overload List | Close Method | Container Enumeration

WebStorage.PutData Method

Upload the data in a byte array buffer to the storage container.

Overload List

Upload the data in a byte array buffer to the storage container.

public bool PutData(string,Attributes,byte[],int);

Upload the data in a MemoryStream object to the storage container.

public bool PutData(string,Attributes,MemoryStream);

Upload the text in a string to the storage container.

public bool PutData(string,Attributes,string);

Upload the data in a byte array buffer to the storage container.

public bool PutData(string,byte[]);

Upload the data in a byte array buffer to the storage container.

public bool PutData(string,byte[],int);

Upload the data in a MemoryStream object to the storage container.

public bool PutData(string,MemoryStream);

Upload the text in a string to the storage container.

public bool PutData(string,string);

Upload the data in a byte array buffer to the storage container.

public bool PutData(string,string,Attributes,byte[],int);

Upload the data in a MemoryStream object to the storage container.

public bool PutData(string,string,Attributes,MemoryStream);

Upload the text in a string to the storage container.

public bool PutData(string,string,Attributes,string);

Upload the data in a byte array buffer to the storage container.

public bool PutData(string,string,byte[],int);

Upload the data in a MemoryStream object to the storage container.

public bool PutData(string,string,MemoryStream);

Upload the text in a string to the storage container.

public bool PutData(string,string,string);

See Also

WebStorage Class | SocketTools Namespace | Close Method | GetData Method | Open Method | Attributes Enumeration

WebStorage.PutData Method (String, String, Attributes, Byte[], Int32)

Upload the data in a byte array buffer to the storage container.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal labelName As String, _
   ByVal contentType As String, _
   ByVal objAttributes As Attributes, _
   ByVal byteBuffer As Byte(), _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool PutData(
   string labelName,
   string contentType,
   Attributes objAttributes,
   byte[] byteBuffer,
   int length
);
```

Parameters

labelName
: A string which specifies the label of the object that should be uploaded to the server. If an object with the same label already exists, it will be replaced with the contents of the buffer

contentType
: An string which identifies the contents of the buffer being stored. If this parameter is an empty string, the method will attempt to automatically determine the content type based on the object label and the contents of the buffer

objAttributes
: An Attributes enumeration which specifies the attributes for the object being stored.

byteBuffer
: A byte array which contains the data to be stored.

length
: An integer value which specifies the number of bytes of data to be stored. This value cannot exceed the size of the byte array.

Return Value

This method returns a Boolean value. If the data has been uploaded to the storage server, it will return **true**, otherwise it will return **false**.

Remarks

The **PutData** method uploads the contents of a byte array to the current storage container. If an object exists with the same label, it will be replaced. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Although storage object labels are similar to Windows file names, they are case-sensitive. When requesting information about an object, your application must specify the label name exactly as it was created using this method. The object label cannot contain wildcard characters.

If the label identifies an object that already exists in the container, and that object was created with the

Attributes.objectReadOnly attribute, this method will fail. To replace a read-only object, the application must explicitly move, rename or delete the existing object.

If a content type is provided, it must specify a valid MIME media type and subtype. For example, normal text has a content type of **text/plain** while XML-formatted text would have a content type of **text/xml**. Data that contains unstructured binary data is typically identified as **application/octet-stream.**. If you do not specify a content type, an appropriate content type will be determined automatically based on the label and the contents of the buffer.

If you wish to upload the contents of a file, use the **PutFile** method.

When this method returns, various properties such as **ObjectId** and **ObjectLabel** will be updated to reflect the values associated with the object that was created or replaced on the server.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.PutData Overload List | Close Method | GetData Method | Open Method | Attributes Enumeration

WebStorage.PutData Method (String, String, Byte[], Int32)

Upload the data in a byte array buffer to the storage container.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal labelName As String, _
   ByVal contentType As String, _
   ByVal byteBuffer As Byte(), _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool PutData(
   string labelName,
   string contentType,
   byte[] byteBuffer,
   int length
);
```

Parameters

labelName
A string which specifies the label of the object that should be uploaded to the server. If an object with the same label already exists, it will be replaced with the contents of the buffer

contentType
An string which identifies the contents of the buffer being stored. If this parameter is an empty string, the method will attempt to automatically determine the content type based on the object label and the contents of the buffer

byteBuffer
A byte array which contains the data to be stored.

length
An integer value which specifies the number of bytes of data to be stored. This value cannot exceed the size of the byte array.

Return Value

This method returns a Boolean value. If the data has been uploaded to the storage server, it will return **true**, otherwise it will return **false**.

Remarks

The **PutData** method uploads the contents of a byte array to the current storage container with default attributes. If an object exists with the same label, it will be replaced. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Although storage object labels are similar to Windows file names, they are case-sensitive. When requesting information about an object, your application must specify the label name exactly as it was created using this method. The object label cannot contain wildcard characters.

If the label identifies an object that already exists in the container, and that object was created with the **Attributes.objectReadOnly** attribute, this method will fail. To replace a read-only object, the application must explicitly move, rename or delete the existing object.

If a content type is provided, it must specify a valid MIME media type and subtype. For example, normal text has a content type of **text/plain** while XML-formatted text would have a content type of **text/xml**. Data that contains unstructured binary data is typically identified as **application/octet-stream.**. If you do not specify a content type, an appropriate content type will be determined automatically based on the label and the contents of the buffer.

If you wish to upload the contents of a file, use the **PutFile** method.

When this method returns, various properties such as **ObjectId** and **ObjectLabel** will be updated to reflect the values associated with the object that was created or replaced on the server.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.PutData Overload List | Close Method | GetData Method | Open Method

WebStorage.PutData Method (String, Attributes, Byte[], Int32)

Upload the data in a byte array buffer to the storage container.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal labelName As String, _
   ByVal objAttributes As Attributes, _
   ByVal byteBuffer As Byte(), _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool PutData(
   string labelName,
   Attributes objAttributes,
   byte[] byteBuffer,
   int length
);
```

Parameters

labelName
A string which specifies the label of the object that should be uploaded to the server. If an object with the same label already exists, it will be replaced with the contents of the buffer

objAttributes
An Attributes enumeration which specifies the attributes for the object being stored.

byteBuffer
A byte array which contains the data to be stored.

length
An integer value which specifies the number of bytes of data to be stored. This value cannot exceed the size of the byte array.

Return Value

This method returns a Boolean value. If the data has been uploaded to the storage server, it will return **true**, otherwise it will return **false**.

Remarks

The **PutData** method uploads the contents of a byte array to the current storage container with a default content type. If an object exists with the same label, it will be replaced. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Although storage object labels are similar to Windows file names, they are case-sensitive. When requesting information about an object, your application must specify the label name exactly as it was created using this method. The object label cannot contain wildcard characters.

If the label identifies an object that already exists in the container, and that object was created with the **Attributes.objectReadOnly** attribute, this method will fail. To replace a read-only object, the application must explicitly move, rename or delete the existing object.

An appropriate content type for the data will be determined automatically based on the label and the contents of the buffer.

If you wish to upload the contents of a file, use the **PutFile** method.

When this method returns, various properties such as **ObjectId** and **ObjectLabel** will be updated to reflect the values associated with the object that was created or replaced on the server.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.PutData Overload List | Close Method | GetData Method | Open Method | Attributes Enumeration

WebStorage.PutData Method (String, Byte[], Int32)

Upload the data in a byte array buffer to the storage container.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal labelName As String, _
   ByVal byteBuffer As Byte(), _
   ByVal length As Integer _
) As Boolean
```

[C#]

```
public bool PutData(
   string labelName,
   byte[] byteBuffer,
   int length
);
```

Parameters

labelName

A string which specifies the label of the object that should be uploaded to the server. If an object with the same label already exists, it will be replaced with the contents of the buffer

byteBuffer

A byte array which contains the data to be stored.

length

An integer value which specifies the number of bytes of data to be stored. This value cannot exceed the size of the byte array.

Return Value

This method returns a Boolean value. If the data has been uploaded to the storage server, it will return **true**, otherwise it will return **false**.

Remarks

The **PutData** method uploads the contents of a byte array to the current storage container with default attributes and a default content type. If an object exists with the same label, it will be replaced. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Although storage object labels are similar to Windows file names, they are case-sensitive. When requesting information about an object, your application must specify the label name exactly as it was created using this method. The object label cannot contain wildcard characters.

If the label identifies an object that already exists in the container, and that object was created with the **Attributes.objectReadOnly** attribute, this method will fail. To replace a read-only object, the application must explicitly move, rename or delete the existing object.

An appropriate content type for the data will be determined automatically based on the label and the contents of the buffer.

If you wish to upload the contents of a file, use the **PutFile** method.

When this method returns, various properties such as **ObjectId** and **ObjectLabel** will be updated to reflect the values associated with the object that was created or replaced on the server.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.PutData Overload List | Close Method | GetData Method | Open Method

WebStorage.PutData Method (String, Byte[])

Upload the data in a byte array buffer to the storage container.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal labelName As String, _
   ByVal byteBuffer As Byte() _
) As Boolean
```

[C#]

```
public bool PutData(
   string labelName,
   byte[] byteBuffer
);
```

Parameters

labelName

A string which specifies the label of the object that should be uploaded to the server. If an object with the same label already exists, it will be replaced with the contents of the buffer

byteBuffer

A byte array which contains the data to be stored.

Return Value

This method returns a Boolean value. If the data has been uploaded to the storage server, it will return **true**, otherwise it will return **false**.

Remarks

The **PutData** method uploads the contents of a byte array to the current storage container with default attributes and a default content type. If an object exists with the same label, it will be replaced. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Although storage object labels are similar to Windows file names, they are case-sensitive. When requesting information about an object, your application must specify the label name exactly as it was created using this method. The object label cannot contain wildcard characters.

If the label identifies an object that already exists in the container, and that object was created with the **Attributes.objectReadOnly** attribute, this method will fail. To replace a read-only object, the application must explicitly move, rename or delete the existing object.

An appropriate content type for the data will be determined automatically based on the label and the contents of the buffer.

If you wish to upload the contents of a file, use the **PutFile** method.

When this method returns, various properties such as **ObjectId** and **ObjectLabel** will be updated to reflect the values associated with the object that was created or replaced on the server.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.PutData Overload List | Close Method | GetData Method | Open Method

WebStorage.PutData Method (String, String, Attributes, MemoryStream)

Upload the data in a MemoryStream object to the storage container.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal labelName As String, _
   ByVal contentType As String, _
   ByVal objAttributes As Attributes, _
   ByVal memStream As MemoryStream _
) As Boolean
```

[C#]

```
public bool PutData(
   string labelName,
   string contentType,
   Attributes objAttributes,
   MemoryStream memStream
);
```

Parameters

labelName
A string which specifies the label of the object that should be uploaded to the server. If an object with the same label already exists, it will be replaced with the contents of the buffer

contentType
An string which identifies the contents of the buffer being stored. If this parameter is an empty string, the method will attempt to automatically determine the content type based on the object label and the contents of the buffer

objAttributes
An Attributes enumeration which specifies the attributes for the object being stored.

memStream
A MemoryStream object which contains the data to be stored. The stream must be open and readable.

Return Value

This method returns a Boolean value. If the data has been uploaded to the storage server, it will return **true**, otherwise it will return **false**.

Remarks

The **PutData** method uploads the contents of a **MemoryStream** object to the current storage container. If a storage object exists with the same label, it will be replaced. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Although storage object labels are similar to Windows file names, they are case-sensitive. When requesting information about an object, your application must specify the label name exactly as it was created using this method. The object label cannot contain wildcard characters.

If the label identifies an object that already exists in the container, and that object was created with the **Attributes.objectReadOnly** attribute, this method will fail. To replace a read-only object, the application must explicitly move, rename or delete the existing object.

If a content type is provided, it must specify a valid MIME media type and subtype. For example, normal text has a content type of **text/plain** while XML-formatted text would have a content type of **text/xml**. Data that contains unstructured binary data is typically identified as **application/octet-stream.**. If you do not specify a content type, an appropriate content type will be determined automatically based on the label and the contents of the buffer.

If you wish to upload the contents of a file, use the **PutFile** method.

When this method returns, various properties such as **ObjectId** and **ObjectLabel** will be updated to reflect the values associated with the object that was created or replaced on the server.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.PutData Overload List | Close Method | GetData Method | Open Method | Attributes Enumeration

WebStorage.PutData Method (String, String, MemoryStream)

Upload the data in a MemoryStream object to the storage container.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal labelName As String, _
   ByVal contentType As String, _
   ByVal memStream As MemoryStream _
) As Boolean
```

[C#]

```
public bool PutData(
   string labelName,
   string contentType,
   MemoryStream memStream
);
```

Parameters

labelName

A string which specifies the label of the object that should be uploaded to the server. If an object with the same label already exists, it will be replaced with the contents of the buffer

contentType

An string which identifies the contents of the buffer being stored. If this parameter is an empty string, the method will attempt to automatically determine the content type based on the object label and the contents of the buffer

memStream

A MemoryStream object which contains the data to be stored. The stream must be open and readable.

Return Value

This method returns a Boolean value. If the data has been uploaded to the storage server, it will return **true**, otherwise it will return **false**.

Remarks

The **PutData** method uploads the contents of a **MemoryStream** object to the current storage container using default attributes. If a storage object exists with the same label, it will be replaced. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Although storage object labels are similar to Windows file names, they are case-sensitive. When requesting information about an object, your application must specify the label name exactly as it was created using this method. The object label cannot contain wildcard characters.

If the label identifies an object that already exists in the container, and that object was created with the **Attributes.objectReadOnly** attribute, this method will fail. To replace a read-only object, the application must explicitly move, rename or delete the existing object.

If a content type is provided, it must specify a valid MIME media type and subtype. For example, normal text has a content type of **text/plain** while XML-formatted text would have a content type of **text/xml**. Data that contains unstructured binary data is typically identified as **application/octet-stream.**. If you do not specify a content type, an appropriate content type will be determined automatically based on the label and the contents of the buffer.

If you wish to upload the contents of a file, use the **PutFile** method.

When this method returns, various properties such as **ObjectId** and **ObjectLabel** will be updated to reflect the values associated with the object that was created or replaced on the server.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.PutData Overload List | Close Method | GetData Method | Open Method

WebStorage.PutData Method (String, Attributes, MemoryStream)

Upload the data in a MemoryStream object to the storage container.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal labelName As String, _
   ByVal objAttributes As Attributes, _
   ByVal memStream As MemoryStream _
) As Boolean
```

[C#]

```
public bool PutData(
   string labelName,
   Attributes objAttributes,
   MemoryStream memStream
);
```

Parameters

labelName

A string which specifies the label of the object that should be uploaded to the server. If an object with the same label already exists, it will be replaced with the contents of the buffer

objAttributes

An Attributes enumeration which specifies the attributes for the object being stored.

memStream

A MemoryStream object which contains the data to be stored. The stream must be open and readable.

Return Value

This method returns a Boolean value. If the data has been uploaded to the storage server, it will return **true**, otherwise it will return **false**.

Remarks

The **PutData** method uploads the contents of a **MemoryStream** object to the current storage container using the default content type for the data being stored. If a storage object exists with the same label, it will be replaced. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Although storage object labels are similar to Windows file names, they are case-sensitive. When requesting information about an object, your application must specify the label name exactly as it was created using this method. The object label cannot contain wildcard characters.

If the label identifies an object that already exists in the container, and that object was created with the **Attributes.objectReadOnly** attribute, this method will fail. To replace a read-only object, the application must explicitly move, rename or delete the existing object.

An appropriate content type is be determined automatically based on the label and the contents of the buffer.

If you wish to upload the contents of a file, use the **PutFile** method.

When this method returns, various properties such as **ObjectId** and **ObjectLabel** will be updated to reflect the values associated with the object that was created or replaced on the server.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.PutData Overload List | Close Method | GetData Method | Open Method | Attributes Enumeration

WebStorage.PutData Method (String, MemoryStream)

Upload the data in a MemoryStream object to the storage container.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal labelName As String, _
   ByVal memStream As MemoryStream _
) As Boolean
```

[C#]

```
public bool PutData(
   string labelName,
   MemoryStream memStream
);
```

Parameters

labelName

A string which specifies the label of the object that should be uploaded to the server. If an object with the same label already exists, it will be replaced with the contents of the buffer

memStream

A MemoryStream object which contains the data to be stored. The stream must be open and readable.

Return Value

This method returns a Boolean value. If the data has been uploaded to the storage server, it will return **true**, otherwise it will return **false**.

Remarks

The **PutData** method uploads the contents of a **MemoryStream** object to the current storage container using the default attributes and content type for the data being stored. If a storage object exists with the same label, it will be replaced. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Although storage object labels are similar to Windows file names, they are case-sensitive. When requesting information about an object, your application must specify the label name exactly as it was created using this method. The object label cannot contain wildcard characters.

If the label identifies an object that already exists in the container, and that object was created with the **Attributes.objectReadOnly** attribute, this method will fail. To replace a read-only object, the application must explicitly move, rename or delete the existing object.

An appropriate content type is be determined automatically based on the label and the contents of the buffer.

If you wish to upload the contents of a file, use the **PutFile** method.

When this method returns, various properties such as **ObjectId** and **ObjectLabel** will be updated to reflect the values associated with the object that was created or replaced on the server.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.PutData Overload List | Close Method | GetData Method | Open Method

WebStorage.PutData Method (String, String, Attributes, String)

Upload the text in a string to the storage container.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal labelName As String, _
   ByVal contentType As String, _
   ByVal objAttributes As Attributes, _
   ByVal textBuffer As String _
) As Boolean
```

[C#]

```
public bool PutData(
   string labelName,
   string contentType,
   Attributes objAttributes,
   string textBuffer
);
```

Parameters

labelName
: A string which specifies the label of the object that should be uploaded to the server. If an object with the same label already exists, it will be replaced with the contents of the buffer

contentType
: An string which identifies the contents of the buffer being stored. If this parameter is an empty string, the method will attempt to automatically determine the content type based on the object label and the contents of the buffer

objAttributes
: An Attributes enumeration which specifies the attributes for the object being stored.

textBuffer
: A string which contains the text to be stored.

Return Value

This method returns a Boolean value. If the data has been uploaded to the storage server, it will return **true**, otherwise it will return **false**.

Remarks

The **PutData** method uploads the contents of a string to the current storage container. If the string contains non-ASCII characters, they will automatically converted to use UTF-8 encoding prior to being stored on the server. If a storage object exists with the same label, it will be replaced. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Although storage object labels are similar to Windows file names, they are case-sensitive. When requesting information about an object, your application must specify the label name exactly as it was created using this method. The object label cannot contain wildcard characters.

If the label identifies an object that already exists in the container, and that object was created with the **Attributes.objectReadOnly** attribute, this method will fail. To replace a read-only object, the application must explicitly move, rename or delete the existing object.

If a content type is provided, it must specify a valid MIME media type and subtype. For example, normal text has a content type of **text/plain** while XML-formatted text would have a content type of **text/xml**. If you do not specify a content type, an appropriate content type will be determined automatically based on the label and the contents of the string. In most cases, the content type will default to **text/plain**.

This version of the **PutData** method should never be used to store binary data, particularly data that contains embedded null bytes. You should only use this method to store plain text. If you need to store binary data, use the version of **PutData** which accepts either a byte array or **MemoryStream** object.

If you wish to upload the contents of a file, use the **PutFile** method.

When this method returns, various properties such as **ObjectId** and **ObjectLabel** will be updated to reflect the values associated with the object that was created or replaced on the server.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.PutData Overload List | Close Method | GetData Method | Open Method | Attributes Enumeration

WebStorage.PutData Method (String, String, String)

Upload the text in a string to the storage container.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal labelName As String, _
   ByVal contentType As String, _
   ByVal textBuffer As String _
) As Boolean
```

[C#]

```
public bool PutData(
   string labelName,
   string contentType,
   string textBuffer
);
```

Parameters

labelName

A string which specifies the label of the object that should be uploaded to the server. If an object with the same label already exists, it will be replaced with the contents of the buffer

contentType

An string which identifies the contents of the buffer being stored. If this parameter is an empty string, the method will attempt to automatically determine the content type based on the object label and the contents of the buffer

textBuffer

A string which contains the text to be stored.

Return Value

This method returns a Boolean value. If the data has been uploaded to the storage server, it will return **true**, otherwise it will return **false**.

Remarks

The **PutData** method uploads the contents of a string to the current storage container using default attributes. If the string contains non-ASCII characters, they will be converted to use UTF-8 encoding prior to being stored on the server. If a storage object exists with the same label, it will be replaced. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Although storage object labels are similar to Windows file names, they are case-sensitive. When requesting information about an object, your application must specify the label name exactly as it was created using this method. The object label cannot contain wildcard characters.

If the label identifies an object that already exists in the container, and that object was created with the **Attributes.objectReadOnly** attribute, this method will fail. To replace a read-only object, the application must explicitly move, rename or delete the existing object.

If a content type is provided, it must specify a valid MIME media type and subtype. For example, normal text has a content type of **text/plain** while XML-formatted text would have a content type of **text/xml**. If you do not specify a content type, an appropriate content type will be determined automatically based on the label and the contents of the string. In most cases, the content type will default to **text/plain**.

This version of the **PutData** method should never be used to store binary data, particularly data that contains embedded null. bytes. You should only use this method to store plain text. If you need to store binary data, use the version of **PutData** which accepts either a byte array or **MemoryStream** object.

If you wish to upload the contents of a file, use the **PutFile** method.

When this method returns, various properties such as **ObjectId** and **ObjectLabel** will be updated to reflect the values associated with the object that was created or replaced on the server.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.PutData Overload List | Close Method | GetData Method | Open Method

WebStorage.PutData Method (String, Attributes, String)

Upload the text in a string to the storage container.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal labelName As String, _
   ByVal objAttributes As Attributes, _
   ByVal textBuffer As String _
) As Boolean
```

[C#]

```
public bool PutData(
   string labelName,
   Attributes objAttributes,
   string textBuffer
);
```

Parameters

labelName
A string which specifies the label of the object that should be uploaded to the server. If an object with the same label already exists, it will be replaced with the contents of the buffer

objAttributes
An Attributes enumeration which specifies the attributes for the object being stored.

textBuffer
A string which contains the text to be stored.

Return Value

This method returns a Boolean value. If the data has been uploaded to the storage server, it will return **true**, otherwise it will return **false**.

Remarks

The **PutData** method uploads the contents of a string to the current storage container. If the string contains non-ASCII characters, they will automatically converted to use UTF-8 encoding prior to being stored on the server. If a storage object exists with the same label, it will be replaced. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Although storage object labels are similar to Windows file names, they are case-sensitive. When requesting information about an object, your application must specify the label name exactly as it was created using this method. The object label cannot contain wildcard characters.

If the label identifies an object that already exists in the container, and that object was created with the **Attributes.objectReadOnly** attribute, this method will fail. To replace a read-only object, the application must explicitly move, rename or delete the existing object.

An appropriate content type will be determined automatically based on the label and the contents of the string. In most cases, the content type will default to **text/plain**.

This version of the **PutData** method should never be used to store binary data, particularly data that contains embedded null bytes. You should only use this method to store plain text. If you need to store binary data, use the version of **PutData** which accepts either a byte array or **MemoryStream** object.

If you wish to upload the contents of a file, use the **PutFile** method.

When this method returns, various properties such as **ObjectId** and **ObjectLabel** will be updated to reflect the values associated with the object that was created or replaced on the server.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.PutData Overload List | Close Method | GetData Method | Open Method | Attributes Enumeration

WebStorage.PutData Method (String, String)

Upload the text in a string to the storage container.

[Visual Basic]

```
Overloads Public Function PutData( _
   ByVal labelName As String, _
   ByVal textBuffer As String _
) As Boolean
```

[C#]

```
public bool PutData(
   string labelName,
   string textBuffer
);
```

Parameters

labelName

A string which specifies the label of the object that should be uploaded to the server. If an object with the same label already exists, it will be replaced with the contents of the buffer

textBuffer

A string which contains the text to be stored.

Return Value

This method returns a Boolean value. If the data has been uploaded to the storage server, it will return **true**, otherwise it will return **false**.

Remarks

The **PutData** method uploads the contents of a string to the current storage container using default attributes. If the string contains non-ASCII characters, they will automatically converted to use UTF-8 encoding prior to being stored on the server. If a storage object exists with the same label, it will be replaced. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Although storage object labels are similar to Windows file names, they are case-sensitive. When requesting information about an object, your application must specify the label name exactly as it was created using this method. The object label cannot contain wildcard characters.

If the label identifies an object that already exists in the container, and that object was created with the **Attributes.objectReadOnly** attribute, this method will fail. To replace a read-only object, the application must explicitly move, rename or delete the existing object.

An appropriate content type will be determined automatically based on the label and the contents of the string. In most cases, the content type will default to **text/plain**.

This version of the **PutData** method should never be used to store binary data, particularly data that contains embedded null bytes. You should only use this method to store plain text. If you need to store binary data, use the version of **PutData** which accepts either a byte array or **MemoryStream** object.

If you wish to upload the contents of a file, use the **PutFile** method.

When this method returns, various properties such as **ObjectId** and **ObjectLabel** will be updated to reflect the values associated with the object that was created or replaced on the server.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.PutData Overload List | Close Method | GetData Method | Open Method

WebStorage.PutFile Method

Upload a local file to the current storage container.

Overload List

Upload a local file to the current storage container.

public bool PutFile(string,string);

Upload a local file to the current storage container.

public bool PutFile(string,string,Attributes);

Upload a local file to the current storage container.

public bool PutFile(string,string,string);

Upload a local file to the current storage container.

public bool PutFile(string,string,string,Attributes);

See Also

WebStorage Class | SocketTools Namespace | Close Method | GetFile Method | Open Method

WebStorage.PutFile Method (String, String)

Upload a local file to the current storage container.

[Visual Basic]

```
Overloads Public Function PutFile( _
   ByVal localFile As String, _
   ByVal labelName As String _
) As Boolean
```

[C#]

```
public bool PutFile(
   string localFile,
   string labelName
);
```

Parameters

localFile
: A string which specifies the name of the local file that will be uploaded to the current storage container. If a path is not specified, the file will be opened in the current working directory.

labelName
: A string which specifies the label of the object that should be created. If an object with the same label already exists, it will be replaced with the contents of the file.

Return Value

This method returns a Boolean value. If the data has been uploaded to the storage server, it will return **true**, otherwise it will return **false**.

Remarks

The **PutFile** method uploads the contents of a file to the current storage container. If an object exists with the same label, it will be replaced. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Although storage object labels are similar to Windows file names, they are case-sensitive. When requesting information about an object, your application must specify the label name exactly as it was created using this method. The object label cannot contain wildcard characters.

If the label identifies an object that already exists in the container, and that object was created with the **Attributes.objectReadOnly** attribute, this method will fail. To replace a read-only object, the application must explicitly move, rename or delete the existing object.

An appropriate content type for the stored object will be determined automatically based on the file name and contents.

If you wish to upload the contents of a string or byte array, use the **PutData** method.

When this method returns, various properties such as **ObjectId** and **ObjectLabel** will be updated to reflect the values associated with the object that was created or replaced on the server.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.PutFile Overload List | Close Method | GetFile Method | Open Method

WebStorage.PutFile Method (String, String, Attributes)

Upload a local file to the current storage container.

[Visual Basic]

```
Overloads Public Function PutFile( _
   ByVal localFile As String, _
   ByVal labelName As String, _
   ByVal objAttributes As Attributes _
) As Boolean
```

[C#]

```
public bool PutFile(
   string localFile,
   string labelName,
   Attributes objAttributes
);
```

Parameters

localFile

A string which specifies the name of the local file that will be uploaded to the current storage container. If a path is not specified, the file will be opened in the current working directory.

labelName

A string which specifies the label of the object that should be created. If an object with the same label already exists, it will be replaced with the contents of the file.

objAttributes

An Attributes enumeration which specifies the attributes for the object being created or replaced.

Return Value

This method returns a Boolean value. If the data has been uploaded to the storage server, it will return **true**, otherwise it will return **false**.

Remarks

The **PutFile** method uploads the contents of a file to the current storage container. If an object exists with the same label, it will be replaced. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Although storage object labels are similar to Windows file names, they are case-sensitive. When requesting information about an object, your application must specify the label name exactly as it was created using this method. The object label cannot contain wildcard characters.

If the label identifies an object that already exists in the container, and that object was created with the **Attributes.objectReadOnly** attribute, this method will fail. To replace a read-only object, the application must explicitly move, rename or delete the existing object.

An appropriate content type for the stored object will be determined automatically based on the file name and contents.

If you wish to upload the contents of a string or byte array, use the **PutData** method.

When this method returns, various properties such as **ObjectId** and **ObjectLabel** will be updated to reflect the values associated with the object that was created or replaced on the server.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.PutFile Overload List | Close Method | GetFile Method | Open Method | Attributes Enumeration

WebStorage.PutFile Method (String, String, String)

Upload a local file to the current storage container.

[Visual Basic]

```
Overloads Public Function PutFile( _
   ByVal localFile As String, _
   ByVal labelName As String, _
   ByVal contentType As String _
) As Boolean
```

[C#]

```
public bool PutFile(
   string localFile,
   string labelName,
   string contentType
);
```

Parameters

localFile
A string which specifies the name of the local file that will be uploaded to the current storage container. If a path is not specified, the file will be opened in the current working directory.

labelName
A string which specifies the label of the object that should be created. If an object with the same label already exists, it will be replaced with the contents of the file.

contentType
An string which identifies the contents of the file being uploaded. If this parameter is an empty string, the method will attempt to automatically determine the content type based on the object label and the contents of the file.

Return Value

This method returns a Boolean value. If the data has been uploaded to the storage server, it will return **true**, otherwise it will return **false**.

Remarks

The **PutFile** method uploads the contents of a file to the current storage container. If an object exists with the same label, it will be replaced. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Although storage object labels are similar to Windows file names, they are case-sensitive. When requesting information about an object, your application must specify the label name exactly as it was created using this method. The object label cannot contain wildcard characters.

If the label identifies an object that already exists in the container, and that object was created with the **Attributes.objectReadOnly** attribute, this method will fail. To replace a read-only object, the application must explicitly move, rename or delete the existing object.

If a content type is provided, it must specify a valid MIME media type and subtype. For example, normal text has a content type of **text/plain** while XML-formatted text would have a content type of **text/xml**. Data that contains unstructured binary data is typically identified as **application/octet-stream**. If you do not specify a content type, an appropriate content type will be determined automatically based on the file name and contents.

If you wish to upload the contents of a string or byte array, use the **PutData** method.

When this method returns, various properties such as **ObjectId** and **ObjectLabel** will be updated to reflect the values associated with the object that was created or replaced on the server.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.PutFile Overload List | Close Method | GetFile Method | Open Method

WebStorage.PutFile Method (String, String, String, Attributes)

Upload a local file to the current storage container.

[Visual Basic]

```
Overloads Public Function PutFile( _
   ByVal localFile As String, _
   ByVal labelName As String, _
   ByVal contentType As String, _
   ByVal objAttributes As Attributes _
) As Boolean
```

[C#]

```
public bool PutFile(
   string localFile,
   string labelName,
   string contentType,
   Attributes objAttributes
);
```

Parameters

localFile
A string which specifies the name of the local file that will be uploaded to the current storage container. If a path is not specified, the file will be opened in the current working directory.

labelName
A string which specifies the label of the object that should be created. If an object with the same label already exists, it will be replaced with the contents of the file.

contentType
An string which identifies the contents of the file being uploaded. If this parameter is an empty string, the method will attempt to automatically determine the content type based on the object label and the contents of the file.

objAttributes
An Attributes enumeration which specifies the attributes for the object being created or replaced.

Return Value

This method returns a Boolean value. If the data has been uploaded to the storage server, it will return **true**, otherwise it will return **false**.

Remarks

The **PutFile** method uploads the contents of a file to the current storage container. If an object exists with the same label, it will be replaced. During the transfer, the **OnProgress** event will fire periodically, enabling the application to update any user interface objects such as a progress bar.

Although storage object labels are similar to Windows file names, they are case-sensitive. When requesting information about an object, your application must specify the label name exactly as it was created using this method. The object label cannot contain wildcard characters.

If the label identifies an object that already exists in the container, and that object was created with the **Attributes.objectReadOnly** attribute, this method will fail. To replace a read-only object, the application must explicitly move, rename or delete the existing object.

If a content type is provided, it must specify a valid MIME media type and subtype. For example, normal text has a content type of **text/plain** while XML-formatted text would have a content type of **text/xml**.

Data that contains unstructured binary data is typically identified as **application/octet-stream**. If you do not specify a content type, an appropriate content type will be determined automatically based on the file name and contents.

If you wish to upload the contents of a string or byte array, use the **PutData** method.

When this method returns, various properties such as **ObjectId** and **ObjectLabel** will be updated to reflect the values associated with the object that was created or replaced on the server.

See Also

WebStorage Class | SocketTools Namespace | WebStorage.PutFile Overload List | Close Method | GetFile Method | Open Method | Attributes Enumeration

WebStorage.RegisterId Method

Register a new application identifier with the storage service.

```
[Visual Basic]
Public Function RegisterId( _
   ByVal appId As String _
) As Boolean
```

```
[C#]
public bool RegisterId(
   string appId
);
```

Parameters

appId

A string which identifies the application requesting access. If the application ID contains illegal characters, the method will fail. See the remarks below on the recommended practice for identifying your application.

Return Value

This method returns a Boolean value. If the application ID has been registered, it will return **true**, otherwise it will return **false**.

Remarks

The **RegisterId** method registers an application ID with the server which uniquely identifies the application that is requesting access to the storage container. The ID must only consist of ASCII letters, numbers, the period and underscore character. Whitespace characters and non-ASCII Unicode characters are not permitted. The maximum length of an application ID string is 63 characters.

It is recommended that you use a standard format for the application ID that consists of your company name, application name and optionally a version number. For example:

- MyCompany.MyApplication
- MyCompany.MyApplication.1

It is important to note that with these two example IDs, although they are similar, they reference two different applications. Objects stored using the first ID will not be accessible using the second ID. If you want to store objects that should be shared between all versions of the application, it is recommended that you use the first form, without the version number. If you want to store objects that should only be accessible to a specific version of your application, then it is recommended that you use the second form that includes the version number.

It is safe to call this method with an application ID that was previously registered. If the provided application ID has already been registered, this method will succeed.

If you no longer wish to use an application ID you have previously registered, you can call the **UnregisterId** method. Exercise caution when unregistering an application. This will cause all objects stored using that ID to be deleted by the storage server. Once an application ID has been unregistered, the operation is permanent. Calling **UnregisterId** and then **RegisterId** again using the same ID will force the system to create new access tokens for your application. You will not be able to regain access to the objects that were previously stored using that application ID.

The application ID is intended to be an application defined human-readable string that uniquely identifies your application. If you want to obtain the internal storage ID associated with your application, get the value of the **StorageId** property. The storage ID is a fixed-length string of letters and numbers guaranteed to be unique across all applications that you register.

It is not required for your application to create a unique application ID. Each storage account has a default internal application ID named **SocketTools.Storage.Default**. This default ID is used if an application-defined ID is not provided to the **Open** method. It is intended to identify storage available to all applications that you create.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.Rename Method

Change the label of an existing storage object.

[Visual Basic]

```
Public Function Rename( _
   ByVal oldLabel As String, _
   ByVal newLabel As String _
) As Boolean
```

[C#]

```
public bool Rename(
   string oldLabel,
   string newLabel
);
```

Parameters

oldLabel
A string which specifies the label of the object that should be renamed.

newLabel
A string which specifies the new label for the object being renamed.

Return Value

This method returns a Boolean value. If the stored object was renamed, it will return **true**, otherwise it will return **false**.

Remarks

The **Rename** method is used to change the label of an existing storage object within the current container. If you wish to move an object to a different, use the **Move** method.

This method updates the current object. Various properties such as **ObjectId** and **ObjectLabel** will reflect the values associated with the object which has been renamed.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.Reset Method

Reset the internal state of the class instance, resetting all properties to their default values.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method returns the class instance to its defaults. Property values are initialized to their internal defaults, the current storage container is closed and any handles allocated by the class will be released.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.Uninitialize Method

Uninitialize the class library and release any resources allocated for the current thread.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method terminates any active connection, releases resources allocated for the current thread and unloads the networking library. After this method has been called, no further client operations may be performed until the class instance has been re-initialized.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the class is no longer needed.

See Also

WebStorage Class | SocketTools Namespace | Initialize Method

WebStorage.UnregisterId Method

Unregister a previously registered application identifier.

[Visual Basic]

```
Public Function UnregisterId( _
   ByVal appId As String _
) As Boolean
```

[C#]

```
public bool UnregisterId(
   string appId
);
```

Parameters

appId

A string which specifies the application ID to be deleted. If the application ID is a zero-length string or contains illegal characters, the method will fail.

Return Value

This method returns a Boolean value. If the application ID was deleted, it will return **true**, otherwise it will return **false**.

Remarks

The **UnregisterId** method deletes the internal storage identifier associated with the application ID and revokes all access tokens that were granted for the application. This operation is immediate and permanent.

Exercise caution when using this method. This will permanently delete all objects that were stored for the specified application. Calling **UnregisterId** and then **RegisterId** again using the same ID will force the system to create new access tokens for your application. You will not be able to regain access to the objects that were previously stored using that ID.

This method cannot be used to unregister the default storage application identifier **SocketTools.Storage.Default**. If this ID is specified, the method will fail with an error indicating that the ID is invalid.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.ValidateId Method

Check an application identifier to ensure it is valid and exists.

[Visual Basic]

```
Public Function ValidateId( _
   ByVal appId As String _
) As Boolean
```

[C#]

```
public bool ValidateId(
   string appId
);
```

Parameters

appId

A string which specifies the application ID to be validated. If the application ID is a zero-length string or contains illegal characters, the method will fail.

Return Value

This method returns a Boolean value. If the application ID is valid and exists, it will return **true**, otherwise it will return **false**.

Remarks

The **ValidateId** method is used to determine if the specified application identifier is valid and has been previously registered using the **RegisterId** method. The ID must only consist of ASCII letters, numbers, the period and underscore character. Whitespace characters and non-ASCII Unicode characters are not permitted. The maximum length of an application ID string is 63 characters.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.ValidateLabel Method

Validate an object label to ensure it uses allowed characters.

```
[Visual Basic]
Public Function ValidateLabel( _
   ByVal labelName As String _
) As Boolean
```

```
[C#]
public bool ValidateLabel(
   string labelName
);
```

Parameters

labelName

A string which specifies the object label to be validated. This parameter cannot be an empty string.

Return Value

This method returns a Boolean value. If the object label is valid, it will return **true**, otherwise it will return **false**.

Remarks

Object labels are similar to Windows file names, except they are case-sensitive. The maximum length of a label string is 511 characters. Leading and trailing whitespace (spaces, tabs, linebreaks, etc.) are ignored in label names.

Illegal characters include ASCII and Unicode control characters 0 through 31, single quotes (39), double quotes (34), less than symbol (60), greater than symbol (62), pipe (124), asterisk (42) and question mark (63). It is not possible to embed null characters in the label name.

Label names may contain forward slash (47) characters and backslash (92) characters, however it is important to note that objects are not stored in a hierarchical structure. An application can create its own folder-like structure to the labels it creates, but this structure is not imposed or enforced by the class.

Labels can contain Unicode characters which are internally encoded as UTF-8.

See Also

WebStorage Class | SocketTools Namespace

WebStorage Events

The events of the **WebStorage** class are listed below. For a complete list of **WebStorage** class members, see the WebStorage Members topic.

Public Instance Events

OnCancel	Occurs when a storage operation is canceled.
OnDownload	Occurs when a storage object has been downloaded.
OnError	Occurs when an storage operation fails.
OnProgress	Occurs as a data stream is being read or written to the client.
OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
OnUpload	Occurs when a storage object has been uploaded.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.OnCancel Event

Occurs when a storage operation is canceled.

[Visual Basic]
```
Public Event OnCancel As EventHandler
```

[C#]
```
public event EventHandler OnCancel;
```

Remarks

The **OnCancel** event is generated when a storage operation, such as uploading or download data, is canceled with the **Cancel** method.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.OnDownload Event

Occurs when a storage object has been downloaded.

[Visual Basic]
```
Public Event OnDownload As OnDownloadEventHandler
```

[C#]
```
public event OnDownloadEventHandler OnDownload;
```

Event Data

The event handler receives an argument of type WebStorage.DownloadEventArgs containing data related to this event. The following **WebStorage.DownloadEventArgs** properties provide information specific to this event.

Property	Description
ObjectLabel	Gets a value which specifies the object label.
ObjectSize	Get a value which specifies the size of the object.

Remarks

The **OnDownload** event is generated when a stored object has been successfully uploaded using either the **GetData** or **GetFile** methods.

When this event occurs, the transfer has completed successfully and the downloaded object becomes the current object for the session. Prior to this event, the **OnProgress** event will occur periodically which provides information about the progress of the data transfer.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.DownloadEventArgs Class

Provides data for the OnDownload event.

For a list of all members of this type, see WebStorage.DownloadEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.WebStorage.DownloadEventArgs

[Visual Basic]
```
Public Class WebStorage.DownloadEventArgs
    Inherits EventArgs
```

[C#]
```
public class WebStorage.DownloadEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

DownloadEventArgs specifies the label and size for for the object that has been downloaded.

The OnDownload event occurs whenever a storage object has been successfully downloaded.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WebStorage (in SocketTools.WebStorage.dll)

See Also

WebStorage.DownloadEventArgs Members | SocketTools Namespace

WebStorage.DownloadEventArgs Members

WebStorage.DownloadEventArgs overview

Public Instance Constructors

WebStorage.DownloadEventArgs Constructor	Initializes a new instance of the WebStorage.DownloadEventArgs class.

Public Instance Properties

ObjectLabel	Gets a value which specifies the object label.
ObjectSize	Get a value which specifies the size of the object.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

WebStorage.DownloadEventArgs Class | SocketTools Namespace

WebStorage.DownloadEventArgs Constructor

Initializes a new instance of the WebStorage.DownloadEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public WebStorage.DownloadEventArgs();
```

See Also

WebStorage.DownloadEventArgs Class | SocketTools Namespace

WebStorage.DownloadEventArgs Properties

The properties of the **WebStorage.DownloadEventArgs** class are listed below. For a complete list of **WebStorage.DownloadEventArgs** class members, see the WebStorage.DownloadEventArgs Members topic.

Public Instance Properties

ObjectLabel	Gets a value which specifies the object label.
ObjectSize	Get a value which specifies the size of the object.

See Also

WebStorage.DownloadEventArgs Class | SocketTools Namespace

WebStorage.DownloadEventArgs.ObjectLabel Property

Gets a value which specifies the object label.

[Visual Basic]
```
Public ReadOnly Property ObjectLabel As String
```

[C#]
```
public string ObjectLabel {get;}
```

Property Value

A string value which contains the object label.

Remarks

This string contains the label of the stored object that was downloaded.

See Also

WebStorage.DownloadEventArgs Class | SocketTools Namespace

WebStorage.DownloadEventArgs.ObjectSize Property

Get a value which specifies the size of the object.

[Visual Basic]

```
Public ReadOnly Property ObjectSize As Long
```

[C#]

```
public long ObjectSize {get;}
```

Property Value

A long integer value which contains the object size in bytes.

Remarks

This integer contains the size of the downloaded object in bytes.

See Also

WebStorage.DownloadEventArgs Class | SocketTools Namespace

WebStorage.OnError Event

Occurs when an storage operation fails.

[Visual Basic]
```
Public Event OnError As OnErrorEventHandler
```

[C#]
```
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type WebStorage.ErrorEventArgs containing data related to this event. The following **WebStorage.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a storage operation fails.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see WebStorage.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.WebStorage.ErrorEventArgs

[Visual Basic]

```
Public Class WebStorage.ErrorEventArgs
    Inherits EventArgs
```

[C#]

```
public class WebStorage.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WebStorage (in SocketTools.WebStorage.dll)

See Also

WebStorage.ErrorEventArgs Members | SocketTools Namespace

WebStorage.ErrorEventArgs Members

WebStorage.ErrorEventArgs overview

Public Instance Constructors

WebStorage.ErrorEventArgs Constructor	Initializes a new instance of the WebStorage.ErrorEventArgs class.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

WebStorage.ErrorEventArgs Class | SocketTools Namespace

WebStorage.ErrorEventArgs Constructor

Initializes a new instance of the WebStorage.ErrorEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public WebStorage.ErrorEventArgs();
```

See Also

WebStorage.ErrorEventArgs Class | SocketTools Namespace

WebStorage.ErrorEventArgs Properties

The properties of the **WebStorage.ErrorEventArgs** class are listed below. For a complete list of **WebStorage.ErrorEventArgs** class members, see the WebStorage.ErrorEventArgs Members topic.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

WebStorage.ErrorEventArgs Class | SocketTools Namespace

WebStorage.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

WebStorage.ErrorEventArgs Class | SocketTools Namespace | Error Property

WebStorage.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public WebStorage.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

WebStorage.ErrorEventArgs Class | SocketTools Namespace | Description Property

WebStorage.OnProgress Event

Occurs as a data stream is being read or written to the client.

[Visual Basic]
```
Public Event OnProgress As OnProgressEventHandler
```

[C#]
```
public event OnProgressEventHandler OnProgress;
```

Event Data

The event handler receives an argument of type WebStorage.ProgressEventArgs containing data related to this event. The following **WebStorage.ProgressEventArgs** properties provide information specific to this event.

Property	Description
BytesCopied	Gets a value which specifies the number of bytes of data that has been sent or received.
BytesTotal	Gets a value which specifies the size of object being transferred.
ObjectLabel	Gets a value which specifies the object label.
Percent	Gets a value which specifies the percentage of data that has been sent or received.

Remarks

The **OnProgress** event occurs as a data stream is being read or written to the client. If large amounts of data are being read or written, this event can be used to update a progress bar or other user-interface component to provide the user with some visual feedback on the progress of the operation.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.ProgressEventArgs Class

Provides data for the OnProgress event.

For a list of all members of this type, see WebStorage.ProgressEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.WebStorage.ProgressEventArgs

[Visual Basic]

```
Public Class WebStorage.ProgressEventArgs
    Inherits EventArgs
```

[C#]

```
public class WebStorage.ProgressEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ProgressEventArgs specifies the number of bytes copied from the data stream, the total number of bytes in the data stream and a completion percentage.

The OnProgress event occurs as a data stream is being read or written to the client. This event only occurs when the **GetData, GetFile**, **PostData, PostFile, PutData, PutFile** or **SubmitForm** methods are called.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WebStorage (in SocketTools.WebStorage.dll)

See Also

WebStorage.ProgressEventArgs Members | SocketTools Namespace

WebStorage.ProgressEventArgs Members

WebStorage.ProgressEventArgs overview

Public Instance Constructors

WebStorage.ProgressEventArgs Constructor	Initializes a new instance of the WebStorage.ProgressEventArgs class.

Public Instance Properties

BytesCopied	Gets a value which specifies the number of bytes of data that has been sent or received.
BytesTotal	Gets a value which specifies the size of object being transferred.
ObjectLabel	Gets a value which specifies the object label.
Percent	Gets a value which specifies the percentage of data that has been sent or received.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

WebStorage.ProgressEventArgs Class | SocketTools Namespace

WebStorage.ProgressEventArgs Constructor

Initializes a new instance of the WebStorage.ProgressEventArgs class.

[Visual Basic]
```
Public Sub New()
```

[C#]
```
public WebStorage.ProgressEventArgs();
```

See Also

WebStorage.ProgressEventArgs Class | SocketTools Namespace

WebStorage.ProgressEventArgs Properties

The properties of the **WebStorage.ProgressEventArgs** class are listed below. For a complete list of **WebStorage.ProgressEventArgs** class members, see the WebStorage.ProgressEventArgs Members topic.

Public Instance Properties

BytesCopied	Gets a value which specifies the number of bytes of data that has been sent or received.
BytesTotal	Gets a value which specifies the size of object being transferred.
ObjectLabel	Gets a value which specifies the object label.
Percent	Gets a value which specifies the percentage of data that has been sent or received.

See Also

WebStorage.ProgressEventArgs Class | SocketTools Namespace

WebStorage.ProgressEventArgs.BytesCopied Property

Gets a value which specifies the number of bytes of data that has been sent or received.

[Visual Basic]

```
Public ReadOnly Property BytesCopied As Long
```

[C#]

```
public long BytesCopied {get;}
```

Property Value

An integer value which specifies the number of bytes of data.

Remarks

The **BytesCopied** property specifies the number of bytes that have been read from the client and stored in the local stream buffer, or written from the stream buffer to the client.

See Also

WebStorage.ProgressEventArgs Class | SocketTools Namespace | BytesTotal Property | Percent Property

WebStorage.ProgressEventArgs.BytesTotal Property

Gets a value which specifies the size of object being transferred.

[Visual Basic]
```
Public ReadOnly Property BytesTotal As Long
```

[C#]
```
public long BytesTotal {get;}
```

Property Value

An integer value which specifies the number of bytes of data.

Remarks

The **BytesTotal** property specifies the total amount of data being exchanged with the storage server.

See Also

WebStorage.ProgressEventArgs Class | SocketTools Namespace | BytesCopied Property | Percent Property

WebStorage.ProgressEventArgs.ObjectLabel Property

Gets a value which specifies the object label.

[Visual Basic]

```
Public ReadOnly Property ObjectLabel As String
```

[C#]

```
public string ObjectLabel {get;}
```

Property Value

A string value which specifies the label for the object being transferred.

See Also

WebStorage.ProgressEventArgs Class | SocketTools Namespace

WebStorage.ProgressEventArgs.Percent Property

Gets a value which specifies the percentage of data that has been sent or received.

[Visual Basic]

```
Public ReadOnly Property Percent As Integer
```

[C#]

```
public int Percent {get;}
```

Property Value

An integer value which specifies a percentage.

Remarks

The **Percent** property specifies the percentage of data that has been transmitted, expressed as an integer value between 0 and 100, inclusive.

See Also

WebStorage.ProgressEventArgs Class | SocketTools Namespace | BytesCopied Property | BytesTotal Property

WebStorage.OnTimeout Event

Occurs when a blocking operation fails to complete before the timeout period elapses.

[Visual Basic]

```
Public Event OnTimeout As EventHandler
```

[C#]

```
public event EventHandler OnTimeout;
```

Remarks

The **OnTimeout** event occurs when a blocking operation, such as sending or receiving data on the client, fails to complete before the specified timeout period elapses. The timeout period for a blocking operation can be adjusted by setting the **Timeout** property.

This event is only generated if the client is in blocking mode.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.OnUpload Event

Occurs when a storage object has been uploaded.

[Visual Basic]
```
Public Event OnUpload As OnUploadEventHandler
```

[C#]
```
public event OnUploadEventHandler OnUpload;
```

Event Data

The event handler receives an argument of type WebStorage.UploadEventArgs containing data related to this event. The following **WebStorage.UploadEventArgs** properties provide information specific to this event.

Property	Description
ObjectLabel	Gets a value which specifies the object label.
ObjectSize	Get a value which specifies the size of the object.

Remarks

The **OnUpload** event is generated when a stored object has been successfully uploaded using either the **PutData** or **PutFile** methods.

When this event occurs, the transfer has completed successfully and the uploaded object becomes the current object for the session. Prior to this event, the **OnProgress** event will occur periodically which provides information about the progress of the data transfer.

See Also

WebStorage Class | SocketTools Namespace

WebStorage.UploadEventArgs Class

Provides data for the OnUpload event.

For a list of all members of this type, see WebStorage.UploadEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.WebStorage.UploadEventArgs

[Visual Basic]

```
Public Class WebStorage.UploadEventArgs
    Inherits EventArgs
```

[C#]

```
public class WebStorage.UploadEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

UploadEventArgs specifies the label and size for for the object that has been uploaded.

The OnUpload event occurs whenever a storage object has been successfully uploaded.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WebStorage (in SocketTools.WebStorage.dll)

See Also

WebStorage.UploadEventArgs Members | SocketTools Namespace

WebStorage.UploadEventArgs Members

WebStorage.UploadEventArgs overview

Public Instance Constructors

WebStorage.UploadEventArgs Constructor	Initializes a new instance of the WebStorage.UploadEventArgs class.

Public Instance Properties

ObjectLabel	Gets a value which specifies the object label.
ObjectSize	Get a value which specifies the size of the object.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

WebStorage.UploadEventArgs Class | SocketTools Namespace

WebStorage.UploadEventArgs Constructor

Initializes a new instance of the WebStorage.UploadEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public WebStorage.UploadEventArgs();
```

See Also

WebStorage.UploadEventArgs Class | SocketTools Namespace

WebStorage.UploadEventArgs Properties

The properties of the **WebStorage.UploadEventArgs** class are listed below. For a complete list of **WebStorage.UploadEventArgs** class members, see the WebStorage.UploadEventArgs Members topic.

Public Instance Properties

ObjectLabel	Gets a value which specifies the object label.
ObjectSize	Get a value which specifies the size of the object.

See Also

WebStorage.UploadEventArgs Class | SocketTools Namespace

WebStorage.UploadEventArgs.ObjectLabel Property

Gets a value which specifies the object label.

[Visual Basic]

```
Public ReadOnly Property ObjectLabel As String
```

[C#]

```
public string ObjectLabel {get;}
```

Property Value

A string value which contains the object label.

Remarks

This string contains the label of the stored object that was uploaded.

See Also

WebStorage.UploadEventArgs Class | SocketTools Namespace

WebStorage.UploadEventArgs.ObjectSize Property

Get a value which specifies the size of the object.

[Visual Basic]

```
Public ReadOnly Property ObjectSize As Long
```

[C#]

```
public long ObjectSize {get;}
```

Property Value

A long integer value which contains the object size in bytes.

Remarks

This integer contains the size of the uploaded object in bytes.

See Also

WebStorage.UploadEventArgs Class | SocketTools Namespace

WebStorage.Attributes Enumeration

The attributes for stored objects.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum WebStorage.Attributes
```

[C#]
```
[Flags]
public enum WebStorage.Attributes
```

Remarks

This identifies the attributes that can be specified for stored objects.

Members

Member Name	Description	Value
objectDefault	Default object attributes. This value is used to indicate the object can be modified, or that the attributes for a previously existing object should not be changed.	0
objectNormal	A normal object that that can be read and modified by the application. This is the default attribute for new objects that are created by the application.	1
objectReadOnly	A read-only object that can only be read by the application. Attempts to modify or replace the contents of the object will fail. Read-only objects can be deleted.	2
objectHidden	A hidden object. Objects with this attribute are not returned when enumerated using the FindFirst and FindNext methods. The object can only be accessed directly when specifying its label.	3

Requirements

Namespace: SocketTools

Assembly: SocketTools.WebStorage (in SocketTools.WebStorage.dll)

See Also

SocketTools Namespace

WebStorage.Container Enumeration

Identifies the storage container type.

[Visual Basic]
```
Public Enum WebStorage.Container
```

[C#]
```
public enum WebStorage.Container
```

Remarks

This enumeration identifies the storage containers that are available to the application. You can think of the storage containers as special folders which store individual objects. In most cases, we recommend using **storageGlobal** which means that stored objects will be accessible to all users of your application.

Members

Member Name	Description
storageDefault	The default storage type. If a storage container has been opened, this value specifies that the current storage type for the container should be used. Otherwise, this value will default to using global storage.
storageGlobal	Global storage. Objects stored using this storage type are available to all users. Any changes made to objects using this storage type will affect all users of the application. Unless there is a specific need to limit access to the objects stored by the application to specific domains, local machines or users, it is recommended that you use this storage type when creating new objects.
storageDomain	Local domain storage. Objects stored using this storage type are only available to users in the same local domain, as defined by the domain name or workgroup name assigned to the local system. If the domain or workgroup name changes, objects previously stored using this storage type will not be available to the application.
storageMachine	Local machine storage. Objects stored using this storage type are only available to users on the same local machine. The local machine is identified by unique characteristics of the system, including the boot volume GUID. Objects previously stored using this storage type will not be available on that system if the boot disk is reformatted.
storageUser	Current user storage. Objects stored using this storage type are only available to the current user logged in on the local machine. The user identifier

	is based on the Windows user SID that is assigned when the user account is created. If the user account is deleted, the objects previously stored using this storage type will not be available to the application.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WebStorage (in SocketTools.WebStorage.dll)

See Also

SocketTools Namespace

WebStorage.ErrorCode Enumeration

Specifies the error codes returned by the WebStorage class.

[Visual Basic]
```
Public Enum WebStorage.ErrorCode
```

[C#]
```
public enum WebStorage.ErrorCode
```

Remarks

The WebStorage class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.
errorProxyAuthenticationRequired	Proxy authentication required.

errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.
errorUnsupportedMediaType	Request specified an unsupported media type.

errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.
errorNoMoreMailboxes	No more mailboxes exist on this server.

errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.
errorInvalidThreadHandle	The specified thread handle is no longer valid.

errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been defined.

errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or unrecognized.

errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.
errorInvalidAccountUuid	The account unique identifier is invalid
errorInvalidAccountId	The application account identifier is invalid
errorInvalidProductId	The product identifier identifier is invalid
errorInvalidSerialNumber	The product serial number is invalid
errorInvalidAppId	The application identifier is invalid
errorInvalidApiKey	The application key is invalid
errorAccountExists	The application account identifier already exists
errorAccountNotCreated	The application account identifier was not created
errorAccountNotFound	The application account identifier was not found
errorAccountNotExpired	access to this account has not expired
errorAccountNotUpdated	The application account could not be updated
errorAccountExpired	access to this account has expired
errorAccountRevoked	access to this account has been revoked
errorApiKeyNotCreated	The application key could not be created
errorApiKeyNotFound	The application key could not be found
errorApiKeyNotExpired	The application key has not expired
errorApiKeyNotUnique	The application key identifier is not unique
errorApiKeyNotUpdated	They application key could not be updated
errorApiKeyNotDeleted	The application key could not be deleted
errorApiKeyExists	The application key already exists
errorApiKeyExpired	The application key has expired and must be refreshed
errorApiKeyRevoked	The application key has been revoked

errorApiKeyAppId	The application was not found or was not specified
errorInvalidToken	The access token is invalid or was not specified
errorTokenNotCreated	The access token could not be created
errorTokenNotFound	The access token could not be found
errorTokenNotExpired	The access token has not expired
errorTokenNotUpdated	The access token was not updated
errorTokenNotDeleted	The access token could not be deleted
errorTokenExpired	The access token has expired and must be refreshed
errorTokenRevoked	The access token has been revoked
errorNoApiKeysFound	no application keys found for this account
errorNoTokensFound	no access tokens found for this application key
errorNoTokensRevoked	no access tokens have been revoked
errorInvalidStorageObject	invalid storage object identifier
errorStorageObjectReadOnly	The storage object is read-only
errorStorageObjectExpired	access to the storage object has expired
errorStorageObjectSize	The storage object size exceeds storage limits
errorStorageObjectDigest	The storage object digest is invalid or cannot be computed
errorStorageObjectExists	a storage object with this label already exists
errorStorageObjectModified	a storage object with this label has been modified
errorStorageObjectNotOwner	The current user is not the storage object owner
errorStorageObjectNotFound	The specified storage object does not exist
errorStorageObjectNotCreated	The storage object was not created
errorStorageObjectNotModified	The storage object was not modified
errorStorageObjectNotRenamed	The storage object was not renamed
errorStorageFolderEmpty	The storage folder does not contain any objects
errorStorageAccountQuota	The storage account has exceeded its quota
errorStorageAccountLimit	The storage account has exceeded its object limit
errorInvalidStorageType	The specified storage type is invalid
errorInvalidStorageProvider	The specified storage provider is not available
errorInvalidStorageRegion	The specified storage region is not available
errorInvalidStorageContainer	The storage container does not exist or cannot be accessed
errorInvalidStorageLabel	The storage object label is invalid or undefined

Requirements

Namespace: SocketTools

Assembly: SocketTools.WebStorage (in SocketTools.WebStorage.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

WebStorage.TraceOptions Enumeration

Specifies the logging options that the WebStorage class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
[Visual Basic]
<Flags>
Public Enum WebStorage.TraceOptions
```

```
[C#]
[Flags]
public enum WebStorage.TraceOptions
```

Remarks

The WebStorage class uses the **TraceOptions** enumeration to specify what kind of debugging information is written to the trace logfile. These options are only meaningful when trace logging is enabled by setting the **Trace** property to **true**.

Members

Member Name	Description	Value
traceDefault	The default trace logging option. This is the same as specifying the **traceInfo** option.	0
traceInfo	All network function calls are written to the trace file. This is the default value.	0
traceError	Only those network function calls which fail are recorded in the trace file.	1
traceWarning	Only those network function calls which fail, or return values which indicate a warning, are recorded in the trace file.	2
traceHexDump	All network function calls are written to the trace file, plus all the data that is sent or received is displayed, in both ASCII and hexadecimal format.	4
traceProcess	All function calls in the current process are logged, rather than only those functions in the current thread. This option is useful for multithreaded applications that are using worker threads.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.WebStorage (in SocketTools.WebStorage.dll)

See Also

SocketTools Namespace

WebStorage.OnDownloadEventHandler Delegate

Represents the method that will handle the OnDownload event.

[Visual Basic]

```
Public Delegate Sub WebStorage.OnDownloadEventHandler( _
   ByVal sender As Object, _
   ByVal e As DownloadEventArgs _
)
```

[C#]

```
public delegate void WebStorage.OnDownloadEventHandler(
      object sender,
      DownloadEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A DownloadEventArgs object that contains the event data.

Remarks

When you create an **OnDownloadEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnDownloadEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WebStorage (in SocketTools.WebStorage.dll)

See Also

SocketTools Namespace

WebStorage.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub WebStorage.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void WebStorage.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ErrorEventArgs that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WebStorage (in SocketTools.WebStorage.dll)

See Also

SocketTools Namespace

WebStorage.OnProgressEventHandler Delegate

Represents the method that will handle the OnProgress event.

[Visual Basic]

```
Public Delegate Sub WebStorage.OnProgressEventHandler( _
   ByVal sender As Object, _
   ByVal e As ProgressEventArgs _
)
```

[C#]

```
public delegate void WebStorage.OnProgressEventHandler(
      object sender,
      ProgressEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A ProgressEventArgs that contains the event data.

Remarks

When you create an **OnProgressEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnProgressEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WebStorage (in SocketTools.WebStorage.dll)

See Also

SocketTools Namespace

WebStorage.OnUploadEventHandler Delegate

Represents the method that will handle the OnUpload event.

[Visual Basic]

```
Public Delegate Sub WebStorage.OnUploadEventHandler( _
   ByVal sender As Object, _
   ByVal e As UploadEventArgs _
)
```

[C#]

```
public delegate void WebStorage.OnUploadEventHandler(
      object sender,
      UploadEventArgs e
   );
```

Parameters

sender

The source of the event.

e

A UploadEventArgs object that contains the event data.

Remarks

When you create an **OnUploadEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnUploadEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WebStorage (in SocketTools.WebStorage.dll)

See Also

SocketTools Namespace

WebStorage.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see WebStorage.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.WebStorage.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class WebStorage.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class WebStorage.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the WebStorage class.

Example

```
<Assembly: SocketTools.WebStorage.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.WebStorage.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.WebStorage (in SocketTools.WebStorage.dll)

See Also

WebStorage.RuntimeLicenseAttribute Members | SocketTools Namespace

WebStorage.RuntimeLicenseAttribute Members

WebStorage.RuntimeLicenseAttribute overview

Public Instance Constructors

WebStorage.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

WebStorage.RuntimeLicenseAttribute Class | SocketTools Namespace

WebStorage.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public WebStorage.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the WebStorage class.

See Also

WebStorage.RuntimeLicenseAttribute Class | SocketTools Namespace

WebStorage.RuntimeLicenseAttribute Properties

The properties of the **WebStorage.RuntimeLicenseAttribute** class are listed below. For a complete list of **WebStorage.RuntimeLicenseAttribute** class members, see the WebStorage.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

WebStorage.RuntimeLicenseAttribute Class | SocketTools Namespace

WebStorage.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

WebStorage.RuntimeLicenseAttribute Class | SocketTools Namespace

WebStorageException Class

The exception that is thrown when a client error occurs.

For a list of all members of this type, see WebStorageException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.WebStorageException

[Visual Basic]

```
Public Class WebStorageException
    Inherits ApplicationException
```

[C#]

```
public class WebStorageException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A WebStorageException is thrown by the WebStorage class when an error occurs.

The default constructor for the WebStorageException class sets the **ErrorCode** property to the last client error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WebStorage (in SocketTools.WebStorage.dll)

See Also

WebStorageException Members | SocketTools Namespace

WebStorageException Members

WebStorageException overview

Public Instance Constructors

WebStorageException	Overloaded. Initializes a new instance of the WebStorageException class.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

WebStorageException Class | SocketTools Namespace

WebStorageException Constructor

Initializes a new instance of the WebStorageException class with the last network error code.

Overload List

Initializes a new instance of the WebStorageException class with the last network error code.

public WebStorageException();

Initializes a new instance of the WebStorageException class with a specified error number.

public WebStorageException(int);

Initializes a new instance of the WebStorageException class with a specified error message.

public WebStorageException(string);

Initializes a new instance of the WebStorageException class with a specified error message and a reference to the inner exception that is the cause of this exception.

public WebStorageException(string,Exception);

See Also

WebStorageException Class | SocketTools Namespace

WebStorageException Constructor ()

Initializes a new instance of the WebStorageException class with the last network error code.

[Visual Basic]

```
Overloads Public Sub New()
```

[C#]

```
public WebStorageException();
```

Remarks

The ctor constructor sets the **ErrorCode** property to the last client error that occurred. For more information about the errors that may occur, refer to the WebStorage.ErrorCode enumeration.

See Also

WebStorageException Class | SocketTools Namespace | WebStorageException Constructor Overload List

WebStorageException Constructor (String)

Initializes a new instance of the WebStorageException class with a specified error message.

[Visual Basic]
```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]
```
public WebStorageException(
   string message
);
```

Parameters

message
The error message that explains the reason for the exception.

Remarks

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

WebStorageException Class | SocketTools Namespace | WebStorageException Constructor Overload List

WebStorageException Constructor (String, Exception)

Initializes a new instance of the WebStorageException class with a specified error message and a reference to the inner exception that is the cause of this exception.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal innerException As Exception _
)
```

[C#]

```
public WebStorageException(
   string message,
   Exception innerException
);
```

Parameters

message

The error message that explains the reason for the exception.

innerException

The exception that is the cause of the current exception. If the *innerException* parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

WebStorageException Class | SocketTools Namespace | WebStorageException Constructor Overload List

WebStorageException Constructor (Int32)

Initializes a new instance of the WebStorageException class with a specified error number.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

[C#]

```
public WebStorageException(
   int code
);
```

Parameters

code

An integer value which specifies an error code.

Remarks

This constructor sets the **ErrorCode** property to the specified error code. For more information about the errors that may occur, refer to the WebStorage.ErrorCode enumeration.

See Also

WebStorageException Class | SocketTools Namespace | WebStorageException Constructor Overload List

WebStorageException Properties

The properties of the **WebStorageException** class are listed below. For a complete list of **WebStorageException** class members, see the WebStorageException Members topic.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

WebStorageException Class | SocketTools Namespace

WebStorageException.ErrorCode Property

Gets a value which specifies the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property ErrorCode As ErrorCode
```

[C#]

```
public WebStorage.ErrorCode ErrorCode {get;}
```

Property Value

Returns a WebStorage.ErrorCode enumeration value which specifies the error code.

Remarks

The **ErrorCode** property returns the error code that specifies the cause of the exception.

The default constructor for the WebStorageException class sets the error code to the last network error that occurred. For more information about the errors that may occur, refer to the WebStorage.ErrorCode enumeration.

See Also

WebStorageException Class | SocketTools Namespace

WebStorageException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]

```
Overrides Public ReadOnly Property Message As String
```

[C#]

```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

WebStorageException Class | SocketTools Namespace

WebStorageException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception. This value is the same as the values returned by the Windows Sockets API. For more information about socket error codes, see the Windows Socket Version 2 API error code documentation in MSDN.

See Also

WebStorageException Class | SocketTools Namespace

WebStorageException Methods

The methods of the **WebStorageException** class are listed below. For a complete list of **WebStorageException** class members, see the WebStorageException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

WebStorageException Class | SocketTools Namespace

WebStorageException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]
```
Overrides Public Function ToString() As String
```

[C#]
```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

WebStorageException Class | SocketTools Namespace

WhoisClient Class

Request registration information for an Internet domain name.

For a list of all members of this type, see WhoisClient Members.

System.Object
  SocketTools.WhoisClient

```
[Visual Basic]
Public Class WhoisClient
    Implements IDisposable
```

```
[C#]
public class WhoisClient : IDisposable
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The WhoisClient class library provides an interface for requesting registration information for an Internet domain name. When a domain name is registered, the organization that registers the domain must provide certain contact information along with technical information such as the primary name servers for that domain.

This class provides an interface for requesting that information and returning it to the program so that it can be displayed or processed. This class would be most commonly used to query the Whois server at **whois.internic.net** to obtain information about a specific Internet domain name or an administrative contact at that domain.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WhoisClient (in SocketTools.WhoisClient.dll)

See Also

WhoisClient Members | SocketTools Namespace

WhoisClient Members

WhoisClient overview

Public Static (Shared) Fields

whoisPortDefault	A constant value which specifies the default port number.
whoisTimeout	A constant value which specifies the default timeout period.

Public Instance Constructors

WhoisClient Constructor	Initializes a new instance of the WhoisClient class.

Public Instance Properties

AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
Blocking	Gets and sets a value which indicates if the client is in blocking mode.
Handle	Gets a value that specifies the client handle allocated for the current session.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsConnected	Gets a value which indicates if a connection to the remote host has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsReadable	Gets a value which indicates if there is data available to be read from the socket connection to the server.
Keyword	Gets and sets the default search keyword.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets the local Internet address that the client is bound to.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.

Options	Gets and sets a value which specifies one or more client options.
RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.
SearchType	Gets and sets the type of query to be performed by the remote server.
Status	Gets a value which specifies the current status of the client.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
Version	Gets a value which returns the current version of the WhoisClient class library.

Public Instance Methods

AttachThread	Attach an instance of the class to the current thread
Cancel	Cancel the current blocking client operation.
Connect	Overloaded. Establish a connection with a remote host.
Disconnect	Terminate the connection with a remote host.
Dispose	Overloaded. Releases all resources used by WhoisClient.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the

	WhoisClient class.
Read	Overloaded. Read data from the server and store it in a byte array.
Reset	Reset the internal state of the object, resetting all properties to their default values.
Search	Overloaded. Search for the specified record using the specified keyword and search type.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.

Public Instance Events

OnCancel	Occurs when a blocking client operation is canceled.
OnConnect	Occurs when a connection is established with the remote host.
OnDisconnect	Occurs when the remote host disconnects from the local system.
OnError	Occurs when an client operation fails.
OnRead	Occurs when data is available to be read from the client.
OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
OnWrite	Occurs when data can be written to the client.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the WhoisClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient Constructor

Initializes a new instance of the WhoisClient class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public WhoisClient();
```

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient Properties

The properties of the **WhoisClient** class are listed below. For a complete list of **WhoisClient** class members, see the WhoisClient Members topic.

Public Instance Properties

AutoResolve	Gets and sets a value that determines if host names and addresses are automatically resolved.
Blocking	Gets and sets a value which indicates if the client is in blocking mode.
Handle	Gets a value that specifies the client handle allocated for the current session.
HostAddress	Gets and sets a value which specifies the Internet address used to establish a connection.
HostName	Gets and sets a value which specifies the host name used to establish a connection.
IsBlocked	Gets a value which indicates if the current thread is performing a blocking client operation.
IsConnected	Gets a value which indicates if a connection to the remote host has been established.
IsInitialized	Gets a value which indicates if the current instance of the class has been initialized successfully.
IsReadable	Gets a value which indicates if there is data available to be read from the socket connection to the server.
Keyword	Gets and sets the default search keyword.
LastError	Gets and sets a value which specifies the last error that has occurred.
LastErrorString	Gets a value which describes the last error that has occurred.
LocalAddress	Gets the local Internet address that the client is bound to.
LocalName	Gets a value which specifies the host name for the local system.
LocalPort	Gets the local port number the client is bound to.
Options	Gets and sets a value which specifies one or more client options.
RemotePort	Gets and sets a value which specifies the remote port number.
RemoteService	Gets and sets a value which specifies the remote service.
SearchType	Gets and sets the type of query to be performed

	by the remote server.
Status	Gets a value which specifies the current status of the client.
ThreadModel	Gets and sets a value which specifies the threading model for the class instance.
ThrowError	Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.
Timeout	Gets and sets a value which specifies a timeout period in seconds.
Trace	Gets and sets a value which indicates if network function logging is enabled.
TraceFile	Gets and sets a value which specifies the name of the logfile.
TraceFlags	Gets and sets a value which specifies the client function tracing flags.
Version	Gets a value which returns the current version of the WhoisClient class library.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.AutoResolve Property

Gets and sets a value that determines if host names and addresses are automatically resolved.

[Visual Basic]

```
Public Property AutoResolve As Boolean
```

[C#]

```
public bool AutoResolve {get; set;}
```

Property Value

Returns **true** if host names are automatically resolved to Internet addresses. The default value is **false**.

Remarks

Setting the **AutoResolve** property determines if the class automatically resolves host names and addresses specified by the **HostName** and **HostAddress** properties. If set to **true**, setting the **HostName** property will cause the class to automatically determine the corresponding IP address and update the **HostAddress** property accordingly. Likewise, setting the **HostAddress** property will cause the class to determine the host name and update the **HostName** property. Setting this property to **false** prevents the class from resolving host names until a connection attempt is made.

It is important to note that setting the **HostName** or **HostAddress** property may cause the current thread to block, sometimes for several seconds, until the name or address is resolved. To prevent this behavior, set this property value to **false**.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.Blocking Property

Gets and sets a value which indicates if the client is in blocking mode.

[Visual Basic]

```
Public Property Blocking As Boolean
```

[C#]

```
public bool Blocking {get; set;}
```

Property Value

Returns **true** if the client is in blocking mode; otherwise it returns **false**. The default value is **true**.

Remarks

Setting the **Blocking** property determines if client operations complete synchronously or asynchronously. If set to **true**, then each client operation (such as sending or receiving data) will return when the operation has completed or timed-out. If set to **false**, client operations will return immediately. If the operation would result in the client blocking (such as attempting to read data when no data has been sent by the remote host), an error is generated.

It is important to note that certain events, such as **OnDisconnect, OnRead** and **OnWrite** are only fired if the client is in non-blocking mode.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.Handle Property

Gets a value that specifies the client handle allocated for the current session.

[Visual Basic]

```
Public ReadOnly Property Handle As Integer
```

[C#]

```
public int Handle {get;}
```

Property Value

An integer which represents a client handle. If there is no active connection, a value of -1 is returned.

Remarks

The **Handle** property specifies the numeric descriptor of the current client session.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.HostAddress Property

Gets and sets a value which specifies the Internet address used to establish a connection.

[Visual Basic]

```
Public Property HostAddress As String
```

[C#]

```
public string HostAddress {get; set;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **HostAddress** property can be used to set the Internet address for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the address is assigned to a valid host name, the **HostName** property will be updated with that value.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.HostName Property

Gets and sets a value which specifies the host name used to establish a connection.

[Visual Basic]

```
Public Property HostName As String
```

[C#]

```
public string HostName {get; set;}
```

Property Value

A string which specifies a host name.

Remarks

The **HostName** property can be used to set the host name for a remote system that you wish to communicate with. If the **AutoResolve** property is set to **true** and the name can be resolved to a valid Internet address, the **HostAddress** property will be updated with that value.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.IsBlocked Property

Gets a value which indicates if the current thread is performing a blocking client operation.

[Visual Basic]

```
Public ReadOnly Property IsBlocked As Boolean
```

[C#]

```
public bool IsBlocked {get;}
```

Property Value

Returns **true** if the current thread is blocking, otherwise returns **false**.

Remarks

The **IsBlocked** property returns **true** if the current thread is blocked performing an operation. Because the Windows Sockets API only permits one blocking operation per thread of execution, this property should be checked before starting any blocking operation in response to an event.

If the **IsBlocked** property returns **false**, this means there are no blocking operations on the current thread at that time. However, this does not guarantee that the next client operation will not fail. An application should always check the return value from a client operation and check the value of the **LastError** property if an error occurs.

Note that this property will return **true** if there is any blocking operation being performed by the current thread, regardless of whether this specific instance of the class is responsible for the blocking operation or not.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.IsConnected Property

Gets a value which indicates if a connection to the remote host has been established.

[Visual Basic]

```
Public ReadOnly Property IsConnected As Boolean
```

[C#]

```
public bool IsConnected {get;}
```

Property Value

Returns **true** if the connection has been established; otherwise returns **false**.

Remarks

The **IsConnected** property can only be used to indicate if there is still a logical connection to the remote host. It cannot be used to detect abnormal conditions such as the remote host aborting the connection, the physical network connection being lost or other critical errors.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.IsInitialized Property

Gets a value which indicates if the current instance of the class has been initialized successfully.

[Visual Basic]

```
Public ReadOnly Property IsInitialized As Boolean
```

[C#]

```
public bool IsInitialized {get;}
```

Property Value

Returns **true** if the class instance has been initialized; otherwise returns **false**.

Remarks

The **IsInitialized** property is used to determine if the current instance of the class has been initialized properly. Normally this is done automatically by the class constructor, however there are circumstances where the class may not be able to initialize itself.

The most common reasons that a class instance may not initialize correctly is that no runtime license key has been defined in the assembly or the license key provided is invalid. It may also indicate a problem with the system configuration or user access rights, such as not being able to load the required networking libraries or not being able to access the system registry.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.IsReadable Property

Gets a value which indicates if there is data available to be read from the socket connection to the server.

[Visual Basic]

```
Public ReadOnly Property IsReadable As Boolean
```

[C#]

```
public bool IsReadable {get;}
```

Property Value

Returns **true** if there is data available to be read; otherwise returns **false**.

Remarks

The **IsReadable** property returns **true** if data can be read from the client without blocking. For non-blocking sessions, this property can be checked before the application attempts to read the client. Note that even if this property does return **true** indicating that there is data available to be read, applications should always check the return value from the **Read** method.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.Keyword Property

Gets and sets the default search keyword.

```
[Visual Basic]
Public Property Keyword As String
```

```
[C#]
public string Keyword {get; set;}
```

Property Value

A string which specifies the default search keyword.

Remarks

The **Keyword** property specifies the value used when querying the remote server. The keyword may refer to a handle, a user name or a mailbox name. Setting this property provides the default keyword for the **Search** method.

Keywords may contain special characters that instruct the server how to match the value. These values are outlined in RFC 954, the standards document that describes the WHOIS/NICNAME protocol. These forms are typically recognized:

Example	Description
value	Search for value as either a user name or a handle.
value...	Search for value that matches anything up to that point.
!value	Search for a handle that matches the given value.
last, first	Search for the specified name.
user@	Search for mailboxes with the given user name.
@host	Search for mailboxes on the specified host.
user@host	Search for mailboxes for the user on the specified host.

If the keyword uses any of these special forms, the **SearchType** property must be set to **searchAny**, which tells the control not to modify the keyword value when submitting the query to the server. Note that all keyword forms may not be supported by a given server, and additional types of searches may be supported.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.LastError Property

Gets and sets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public Property LastError As ErrorCode
```

[C#]

```
public WhoisClient.ErrorCode LastError {get; set;}
```

Property Value

Returns an ErrorCode enumeration value which specifies the last error code.

Remarks

The **LastError** property returns the error code associated with the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

It is possible to explicitly clear the last error code by assigning the property to the value **ErrorCode.errorNone**.

The error code value can be cast to an integer value for display purposes if required. For a description of the error that can be displayed using a message box or some other similar mechanism, get the value of the **LastErrorString** property.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.LastErrorString Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property LastErrorString As String
```

[C#]

```
public string LastErrorString {get;}
```

Property Value

A string which describes the last error that has occurred.

Remarks

The **LastErrorString** property can be used to obtain a description of the last error that occurred for the current instance of the class. It is important to note that this value only has meaning if the previous method indicates that an error has actually occurred.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.LocalAddress Property

Gets the local Internet address that the client is bound to.

[Visual Basic]

```
Public ReadOnly Property LocalAddress As String
```

[C#]

```
public string LocalAddress {get;}
```

Property Value

A string which specifies an Internet address.

Remarks

The **LocalAddress** property returns the local Internet address that the client is bound to when a connection is established with a remote host. This property may return either an IPv4 or IPv6 formatted address, depending on the type of connection that was established.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.LocalName Property

Gets a value which specifies the host name for the local system.

[Visual Basic]
```
Public ReadOnly Property LocalName As String
```

[C#]
```
public string LocalName {get;}
```

Property Value

A string which specifies the local host name.

Remarks

The **LocalName** property returns the fully-qualified host name assigned to the local system. If the system has not been configured with an Internet domain name, then this property will return the NetBIOS name assigned to the local system.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.LocalPort Property

Gets the local port number the client is bound to.

[Visual Basic]

```
Public ReadOnly Property LocalPort As Integer
```

[C#]

```
public int LocalPort {get;}
```

Property Value

An integer value which specifies a port number. The default value is 0.

Remarks

The **LocalPort** property is used to identify the local port number that the client is bound to to when a connection is established with a remote host.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.Options Property

Gets and sets a value which specifies one or more client options.

[Visual Basic]
```
Public Property Options As WhoisOptions
```

[C#]
```
public WhoisClient.WhoisOptions Options {get; set;}
```

Property Value

Returns one or more WhoisOptions enumeration flags which specify the options for the client. The default value for this property is **whoisOptionNone**.

Remarks

The **Options** property specifies one or more default options options which are used when establishing a connection using the **Connect** method.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.RemotePort Property

Gets and sets a value which specifies the remote port number.

[Visual Basic]

```
Public Property RemotePort As Integer
```

[C#]

```
public int RemotePort {get; set;}
```

Property Value

An integer value which specifies a port number.

Remarks

The **RemotePort** property is used to set the port number that will be used to establish a connection with a remote host. If the port number specifies a well-known port, the **RemoteService** property will be updated with that name.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.SearchType Property

Gets and sets the type of query to be performed by the remote server.

[Visual Basic]
```
Public Property SearchType As WhoisSearchType
```

[C#]
```
public WhoisClient.WhoisSearchType SearchType {get; set;}
```

Property Value

A WhoisSearchType enumeration value which specifies the default type of search to be performed.

Remarks

The **SearchType** property specifies the default type of query to be performed by the remote server using the **Search** method. If you wish to perform a more complex query using the syntax outlined in RFC 954, specify a search type of **searchAny** and then provide the search string value that you want to submit.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.Status Property

Gets a value which specifies the current status of the client.

[Visual Basic]
```
Public ReadOnly Property Status As WhoisStatus
```

[C#]
```
public WhoisClient.WhoisStatus Status {get;}
```

Property Value

A WhoisStatus enumeration value which specifies the current client status.

Remarks

The **Status** property returns the current status of the client. This property can be used to check on blocking connections to determine if the client is interacting with the remote host before taking some action.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.ThreadModel Property

Gets and sets a value which specifies the threading model for the class instance.

[Visual Basic]
```
Public Property ThreadModel As ThreadingModel
```

[C#]
```
public WhoisClient.ThreadingModel ThreadModel {get; set;}
```

Property Value

Returns one or more ThreadingModel enumeration value which specifies the threading model for the client. The default value for this property is **modelSingleThread**.

Remarks

The **ThreadModel** property specifies the threading model that is used by the class instance when a connection is established. The default value for this property is **modelSingleThread**, which specifies that only the thread that established the connection should be permitted to invoke methods. It is important to note that this threading model does not limit the application to a single thread of execution. When a session is established using the **Connect** method, that session is attached to the thread that created it. From that point on, until the session is terminated, only the owner may invoke methods in that instance of the class. The ownership of the class instance may be transferred from one thread to another using the **AttachThread** method.

Setting this property to **modelFreeThread** disables certain internal safety checks that are performed by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads. The application must ensure that no two threads will attempt to invoke a blocking method at the same time. In other words, if one thread invokes a method, the application must ensure that another thread will not attempt to invoke any other method at the same time using the same instance of the class.

Changing the value of this property will not affect an active client session. The threading model must be specified prior to invoking the **Connect** method.

See Also

WhoisClient Class | SocketTools Namespace | AttachThread Method | ThreadingModel Enumeration | ThreadModel Attribute

WhoisClient.ThrowError Property

Gets and sets a value which specifies if method calls should throw exceptions when an error occurs.

[Visual Basic]

```
Public Property ThrowError As Boolean
```

[C#]

```
public bool ThrowError {get; set;}
```

Property Value

Returns **true** if method calls will generate exceptions when an error occurs; otherwise returns **false**. The default value is **false**.

Remarks

Error handling for when calling class methods can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to **false**, the application should check the return value of any method that is used, and report errors based upon the documented value of the return code. It is the responsibility of the application to interpret the error code, if it is desired to explain the error in addition to reporting it. This is the default behavior.

If the **ThrowError** property is set to **true**, then exceptions will be generated whenever a method call fails. The program must be written to catch these exceptions and take the appropriate action when an error occurs. Failure to handle an exception will cause the program to terminate abnormally.

Note that if an error occurs while a property is being read or modified, an exception will be raised regardless of the value of the **ThrowError** property.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.Timeout Property

Gets and sets a value which specifies a timeout period in seconds.

[Visual Basic]

```
Public Property Timeout As Integer
```

[C#]

```
public int Timeout {get; set;}
```

Property Value

An integer value which specifies a timeout period in seconds.

Remarks

Setting the **Timeout** property specifies the number of seconds until a blocking operation fails and returns an error.

The timeout period is only used when the client is in blocking mode. Although this property can be changed when the client is in non-blocking mode, the value will be ignored until the client is returned to blocking mode.

For most applications it is recommended the timeout period be set between 10 and 20 seconds.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.Trace Property

Gets and sets a value which indicates if network function logging is enabled.

[Visual Basic]

```
Public Property Trace As Boolean
```

[C#]

```
public bool Trace {get; set;}
```

Property Value

Returns **true** if network function tracing is enabled; otherwise returns **false**. The default value is **false**.

Remarks

The **Trace** property is used to enable (or disable) the tracing of network function calls. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is available in all of the SocketTools networking classes and is enabled or disabled for an entire process. This means that once trace logging is enabled for a given component, all of the function calls made by the process using any of the SocketTools classes will be logged. For example, if you have an application using both the File Transfer Protocol and Post Office Protocol classes, and you set the **Trace** property to **true**, function calls made by both classes will be logged. Additionally, enabling a trace is cumulative, and tracing is not stopped until it is disabled for all classes used by the process.

If trace logging is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the logfile is fairly large. Since logfiles can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

When redistributing your application, make sure that you include the **SocketTools10.TraceLog.dll** module with your installation. If this library is not present, then no trace output will be generated and the value of the **Trace** property will be ignored. Only those function calls made by the SocketTools networking classes will be logged. Calls made directly to the Windows Sockets API, or calls made by other classes, will not be logged.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.TraceFile Property

Gets and sets a value which specifies the name of the logfile.

[Visual Basic]

```
Public Property TraceFile As String
```

[C#]

```
public string TraceFile {get; set;}
```

Property Value

A string which specifies the name of the file.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string (the default value), then a file named **SocketTools.log** is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since network function tracing is enabled per-process, the trace file is shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFile** property should be set to the same value for each instance. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
 MyApp INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0 MyApp WRN:
connect(46, 192.0.0.1:1234, 16) returned -1 [10035] MyApp ERR: accept(46,
NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is appended to the record (the value is placed inside brackets).

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a pointer (a memory address), it is recorded as a hexadecimal value preceded with "0x". A special type of pointer, called a null pointer, is recorded as NULL. Those functions which expect socket addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the control is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Note that if the specified file cannot be created, or the user does not have permission to modify an existing file, the error is silently ignored and no trace output will be generated.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.TraceFlags Property

Gets and sets a value which specifies the client function tracing flags.

[Visual Basic]
```
Public Property TraceFlags As TraceOptions
```

[C#]
```
public WhoisClient.TraceOptions TraceFlags {get; set;}
```

Property Value

A TraceOptions enumeration which specifies the amount of detail written to the trace logfile.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled.

Because network function tracing is enabled per-process, the trace flags are shared by all instances of the class being used. If multiple class instances have tracing enabled, the **TraceFlags** property should be set to the same value for each instance. Changing the trace flags for any one instance of the class will affect the logging performed for all SocketTools classes used by the application.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being written and the error **errorOperationWouldBlock** occurs, a warning is generated because the application simply needs to attempt to write the data at a later time.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.Version Property

Gets a value which returns the current version of the WhoisClient class library.

[Visual Basic]
```
Public ReadOnly Property Version As String
```

[C#]
```
public string Version {get;}
```

Property Value

A string which specifies the version of the class library.

Remarks

The **Version** property returns a string which identifies the current version and build of the WhoisClient class library. This value can be used by an application for validation and debugging purposes.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient Methods

The methods of the **WhoisClient** class are listed below. For a complete list of **WhoisClient** class members, see the WhoisClient Members topic.

Public Instance Methods

AttachThread	Attach an instance of the class to the current thread
Cancel	Cancel the current blocking client operation.
Connect	Overloaded. Establish a connection with a remote host.
Disconnect	Terminate the connection with a remote host.
Dispose	Overloaded. Releases all resources used by WhoisClient.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Initialize	Overloaded. Initialize an instance of the WhoisClient class.
Read	Overloaded. Read data from the server and store it in a byte array.
Reset	Reset the internal state of the object, resetting all properties to their default values.
Search	Overloaded. Search for the specified record using the specified keyword and search type.
ToString (inherited from Object)	Returns a String that represents the current Object.
Uninitialize	Uninitialize the class library and release any resources allocated for the current thread.

Protected Instance Methods

Dispose	Overloaded. Releases the unmanaged resources allocated by the WhoisClient class and optionally releases the managed resources.
Finalize	Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.AttachThread Method

Attach an instance of the class to the current thread

[Visual Basic]
```
Public Function AttachThread() As Boolean
```

[C#]
```
public bool AttachThread();
```

Return Value

A boolean value which specifies if the client could be attached to the current thread. If this method returns **false**, the client could not be attached to the thread and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

When an instance of the class is created it is associated with the current thread that created it. Normally, if another thread attempts to perform an operation using that instance, an error is returned. This is used to ensure that other threads cannot interfere with an operation being performed by the owner thread. In some cases, it may be desirable for one thread in an application to create an instance of the class, establish a connection and then pass that instance to another worker thread. The **AttachThread** method can be used to change the ownership of the class instance to the new worker thread.

This method should be called by the new thread immediately after it has been created, and if the new thread does not release the handle itself, the ownership of the handle should be restored by the original thread. Under no circumstances should **AttachThread** be used to forcibly destroy an instance of a class allocated by another thread while a blocking operation is in progress. To cancel a blocking operation, use the **Cancel** method and then delete the class instance after the blocking function exits and control is returned to the current thread.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.Cancel Method

Cancel the current blocking client operation.

[Visual Basic]

```
Public Sub Cancel()
```

[C#]

```
public void Cancel();
```

Remarks

When the **Cancel** method is called, the blocking client operation will not immediately fail. An internal flag is set which causes the blocking operation to exit with an error. This means that the application cannot cancel an operation and immediately perform some other blocking function. Instead it must allow the calling stack to unwind, returning back to the blocking operation before making any further function calls.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.Connect Method

Establish a connection with a remote host.

Overload List

Establish a connection with a remote host.

public bool Connect();

Establish a connection with a remote host.

public bool Connect(string);

Establish a connection with a remote host.

public bool Connect(string,int);

Establish a connection with a remote host.

public bool Connect(string,int,int);

Establish a connection with a remote host.

public bool Connect(string,int,int,WhoisOptions);

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.Connect Method ()

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect() As Boolean
```

[C#]

```
public bool Connect();
```

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

Remarks

This method establishes a connection using assigned property values.

The value of the **HostName** or **HostAddress** property will be used to determine the host name or address to connect to.

The value of the **RemotePort** property will be used to determine the port number to connect to.

The value of the **Timeout** property will be used to specify the timeout period.

The value of the **Options** property will be used to specify the default options for the connection.

See Also

WhoisClient Class | SocketTools Namespace | WhoisClient.Connect Overload List

WhoisClient.Connect Method (String)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

WhoisClient Class | SocketTools Namespace | WhoisClient.Connect Overload List

WhoisClient.Connect Method (String, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

WhoisClient Class | SocketTools Namespace | WhoisClient.Connect Overload List

WhoisClient.Connect Method (String, Int32, Int32)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   int timeout
);
```

Parameters

hostName

A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort

An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

timeout

An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

WhoisClient Class | SocketTools Namespace | WhoisClient.Connect Overload List

WhoisClient.Connect Method (String, Int32, Int32, WhoisOptions)

Establish a connection with a remote host.

[Visual Basic]

```
Overloads Public Function Connect( _
   ByVal hostName As String, _
   ByVal hostPort As Integer, _
   ByVal timeout As Integer, _
   ByVal options As WhoisOptions _
) As Boolean
```

[C#]

```
public bool Connect(
   string hostName,
   int hostPort,
   int timeout,
   WhoisOptions options
);
```

Parameters

hostName
A string which specifies the remote host to establish a connection with. This may specify a host name or an Internet address in dot-notation.

hostPort
An integer which specifies the port number for the connection. This value must be greater than zero and the maximum valid port number is 65535.

timeout
An integer value that specifies the number of seconds that the method will wait for the connection to complete before failing the operation and returning to the caller. This value is only meaningful for blocking connections.

options
One or more of the WhoisOptions enumeration flags.

Return Value

A boolean value which specifies if the connection has been established. If the client is in blocking mode, a return value of **true** indicates that the connection has completed and the application may send and receive data from the remote host. If the client is in non-blocking mode, a return value of **true** indicates that the client has successfully created a socket and the connection is in progress.

When a non-blocking connection has completed, the OnConnect event will be fired. If this method returns **false**, the connection could not be established and the application should check the value of the LastError property to determine the cause of the failure.

See Also

WhoisClient Class | SocketTools Namespace | WhoisClient.Connect Overload List

WhoisClient.Disconnect Method

Terminate the connection with a remote host.

[Visual Basic]

```
Public Sub Disconnect()
```

[C#]

```
public void Disconnect();
```

Remarks

The **Disconnect** method terminates the connection with the remote host and releases the client handle allocated by the class. Note that the socket is not immediately released when the connection is terminated and will enter a wait state for two minutes. After the time wait period has elapsed, the client will be released by the operating system. This is a normal safety mechanism to handle any packets that may arrive after the connection has been closed.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.Dispose Method

Releases all resources used by WhoisClient.

Overload List

Releases all resources used by WhoisClient.

public void Dispose();

Releases the unmanaged resources allocated by the WhoisClient class and optionally releases the managed resources.

protected virtual void Dispose(bool);

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.Dispose Method ()

Releases all resources used by WhoisClient.

[Visual Basic]

```
NotOverridable Overloads Public Sub Dispose() _
    Implements IDisposable.Dispose
```

[C#]

```
public void Dispose();
```

Implements

IDisposable.Dispose

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

See Also

WhoisClient Class | SocketTools Namespace | WhoisClient.Dispose Overload List

WhoisClient.Dispose Method (Boolean)

Releases the unmanaged resources allocated by the WhoisClient class and optionally releases the managed resources.

[Visual Basic]

```
Overridable Overloads Protected Sub Dispose( _
   ByVal disposing As Boolean _
)
```

[C#]

```
protected virtual void Dispose(
   bool disposing
);
```

Parameters

disposing

A boolean value which should be specified as **true** to release both managed and unmanaged resources; **false** to release only unmanaged resources.

Remarks

The **Dispose** method terminates any active connection and explicitly releases the resources allocated for this instance of the class. In some cases, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. The **Dispose** method provides explicit control over these resources.

Unlike the **Uninitialize** method, once the **Dispose** method has been called, that instance of the class cannot be re-initialized and you should not attempt to access class properties or invoke any methods. Note that this method can be called even if other references to the object are active.

You should call **Dispose** in your derived class when you are finished using the derived class. The **Dispose** method leaves the derived class in an unusable state. After calling **Dispose**, you must release all references to the derived class and the **WhoisClient** class so the memory they were occupying can be reclaimed by garbage collection.

See Also

WhoisClient Class | SocketTools Namespace | WhoisClient.Dispose Overload List

WhoisClient.Finalize Method

Destroys an instance of the class, releasing the resources allocated for the session and unloading the networking library.

[Visual Basic]

```
Overrides Protected Sub Finalize()
```

[C#]

```
protected override void Finalize();
```

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.Initialize Method

Initialize an instance of the WhoisClient class.

Overload List

Initialize an instance of the WhoisClient class.

public bool Initialize();

Initialize an instance of the WhoisClient class.

public bool Initialize(string);

See Also

WhoisClient Class | SocketTools Namespace | Uninitialize Method

WhoisClient.Initialize Method ()

Initialize an instance of the WhoisClient class.

[Visual Basic]

```
Overloads Public Function Initialize() As Boolean
```

[C#]

```
public bool Initialize();
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the WhoisClient class, loading the networking library and allocating resources for the current thread. Typically it is not necessary to explicitly call this method because the instance of the class is initialized by the class constructor. However, if the **Uninitialize** method is called, the class must be re-initialized before any other methods are called.

See Also

WhoisClient Class | SocketTools Namespace | WhoisClient.Initialize Overload List | Uninitialize Method

WhoisClient.Initialize Method (String)

Initialize an instance of the WhoisClient class.

[Visual Basic]

```
Overloads Public Function Initialize( _
   ByVal licenseKey As String _
) As Boolean
```

[C#]

```
public bool Initialize(
   string licenseKey
);
```

Return Value

A boolean value which specifies if the class was initialized successfully.

Remarks

The Initialize method can be used to explicitly initialize an instance of the WhoisClient class, loading the networking library and allocating resources for the current thread. Typically an application would define the license key as a custom attribute, however this method can be used to initialize the class directly.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the WhoisClient class.

Example

The following example shows how to use the Initialize method to initialize an instance of the class. This example assumes that the license key string has been defined in code.

```
SocketTools.WhoisClient whoisClient = new SocketTools.WhoisClient();

if (whoisClient.Initialize(strLicenseKey) == false)
{
    MessageBox.Show(whoisClient.LastErrorString, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    return;
}
```

```
Dim whoisClient As New SocketTools.WhoisClient

If whoisClient.Initialize(strLicenseKey) = False Then
    MsgBox(whoisClient.LastErrorString, vbIconExclamation)
    Exit Sub
End If
```

See Also

WhoisClient Class | SocketTools Namespace | WhoisClient.Initialize Overload List | RuntimeLicenseAttribute Class | Uninitialize Method

WhoisClient.Read Method

Read data from the server and store it in a byte array.

Overload List

Read data from the server and store it in a byte array.

public int Read(byte[]);

Read data from the server and store it in a byte array.

public int Read(byte[],int);

Read data from the server and store it in a string.

public int Read(ref string);

Read data from the server and store it in a string.

public int Read(ref string,int);

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.Read Method (Byte[])

Read data from the server and store it in a byte array.

[Visual Basic]

```
Overloads Public Function Read( _
   ByVal buffer As Byte() _
) As Integer
```

[C#]

```
public int Read(
   byte[] buffer
);
```

Parameters

buffer

A byte array that the data will be stored in.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the size of the byte array passed to the method. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

See Also

WhoisClient Class | SocketTools Namespace | WhoisClient.Read Overload List

WhoisClient.Read Method (Byte[], Int32)

Read data from the server and store it in a byte array.

```
[Visual Basic]
Overloads Public Function Read( _
   ByVal buffer As Byte(), _
   ByVal length As Integer _
) As Integer
```

```
[C#]
public int Read(
   byte[] buffer,
   int length
);
```

Parameters

buffer

A byte array that the data will be stored in.

length

An integer value which specifies the maximum number of bytes of data to read. This value cannot be larger than the size of the buffer specified by the caller.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the number of bytes specified. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

See Also

WhoisClient Class | SocketTools Namespace | WhoisClient.Read Overload List

WhoisClient.Read Method (String)

Read data from the server and store it in a string.

[Visual Basic]

```
Overloads Public Function Read( _
   ByRef buffer As String _
) As Integer
```

[C#]

```
public int Read(
   ref string buffer
);
```

Parameters

buffer
A string that will contain the data read from the client.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to a maximum of 4096 bytes. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

See Also

WhoisClient Class | SocketTools Namespace | WhoisClient.Read Overload List

WhoisClient.Read Method (String, Int32)

Read data from the server and store it in a string.

```
[Visual Basic]
Overloads Public Function Read( _
   ByRef buffer As String, _
   ByVal length As Integer _
) As Integer
```

```
[C#]
public int Read(
   ref string buffer,
   int length
);
```

Parameters

buffer
A string that will contain the data read from the client.

length
An integer value which specifies the maximum number of bytes of data to read.

Return Value

An integer value which specifies the number of bytes actually read from the server. A return value of zero specifies that the remote host has closed the connection and there is no more data available to be read. If an error occurs, a value of -1 is returned and the application should check the value of the **LastError** property to determine the cause of the failure.

Remarks

The **Read** method returns data that has been read from the server, up to the number of bytes specified. If no data is available to be read, an error will be generated if the client is in non-blocking mode. If the client is in blocking mode, the program will stop until data is received from the server or the connection is closed.

This method should only be used if the remote host is sending data that consists of printable characters. Binary data should be read using the method that accepts a byte array as the buffer parameter.

See Also

WhoisClient Class | SocketTools Namespace | WhoisClient.Read Overload List

WhoisClient.Reset Method

Reset the internal state of the object, resetting all properties to their default values.

[Visual Basic]

```
Public Sub Reset()
```

[C#]

```
public void Reset();
```

Remarks

The **Reset** method returns the object to its default state. If a socket has been allocated, it will be released and any active connections will be terminated. All properties will be reset to their default values.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.Search Method

Search for the specified record using the default keyword and search type.

Overload List

Search for the specified record using the default keyword and search type.

public bool Search();

Search for the specified record using the default search type.

public bool Search(string);

Search for the specified record using the specified keyword and search type.

public bool Search(string,WhoisSearchType);

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.Search Method ()

Search for the specified record using the default keyword and search type.

[Visual Basic]

```
Overloads Public Function Search() As Boolean
```

[C#]

```
public bool Search();
```

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Search** method submits the specified keyword and type of search to the server. The data returned by the server can be read using the **Read** method. Note that the text returned by a UNIX based server may only contain linefeeds at the end of each line of text, rather than the standard carriage return/linefeed used on Windows systems.

The value of the **Keyword** property will specify the search keyword. The value of the **SearchType** property is used to specify the type of search that is performed. By default, all record types will be searched.

See Also

WhoisClient Class | SocketTools Namespace | WhoisClient.Search Overload List

WhoisClient.Search Method (String)

Search for the specified record using the default search type.

[Visual Basic]

```
Overloads Public Function Search( _
   ByVal keyword As String _
) As Boolean
```

[C#]

```
public bool Search(
   string keyword
);
```

Parameters

keyword

A string which specifies the keyword to search for. Typically this is the name of a domain, user handle or an email address.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Search** method submits the specified keyword and type of search to the server. The data returned by the server can be read using the **Read** method. Note that the text returned by a UNIX based server may only contain linefeeds at the end of each line of text, rather than the standard carriage return/linefeed used on Windows systems.

The value of the **SearchType** property is used to specify the type of search that is performed. By default, all record types will be searched.

See Also

WhoisClient Class | SocketTools Namespace | WhoisClient.Search Overload List

WhoisClient.Search Method (String, WhoisSearchType)

Search for the specified record using the specified keyword and search type.

[Visual Basic]

```
Overloads Public Function Search( _
   ByVal keyword As String, _
   ByVal search As WhoisSearchType _
) As Boolean
```

[C#]

```
public bool Search(
   string keyword,
   WhoisSearchType search
);
```

Parameters

keyword
: A string which specifies the keyword to search for. Typically this is the name of a domain, user handle or an email address.

search
: A WhoisSearchType enumeration which specifies the type of search to be performed.

Return Value

This method returns a Boolean value. If the method succeeds, the return value is **true**. If the method fails, the return value is **false**. To get extended error information, check the value of the **LastError** property.

Remarks

The **Search** method submits the specified keyword and type of search to the server. The data returned by the server can be read using the **Read** method. Note that the text returned by a UNIX based server may only contain linefeeds at the end of each line of text, rather than the standard carriage return/linefeed used on Windows systems.

See Also

WhoisClient Class | SocketTools Namespace | WhoisClient.Search Overload List

WhoisClient.Uninitialize Method

Uninitialize the class library and release any resources allocated for the current thread.

[Visual Basic]

```
Public Sub Uninitialize()
```

[C#]

```
public void Uninitialize();
```

Remarks

The **Uninitialize** method terminates any active connection, releases resources allocated for the current thread and unloads the networking library. After this method has been called, no further client operations may be performed until the class instance has been re-initialized.

If the **Initialize** method is explicitly called by the application, it should be matched by a call to the **Uninitialize** method when that instance of the class is no longer needed.

See Also

WhoisClient Class | SocketTools Namespace | Initialize Method

WhoisClient Events

The events of the **WhoisClient** class are listed below. For a complete list of **WhoisClient** class members, see the WhoisClient Members topic.

Public Instance Events

⚡ OnCancel	Occurs when a blocking client operation is canceled.
⚡ OnConnect	Occurs when a connection is established with the remote host.
⚡ OnDisconnect	Occurs when the remote host disconnects from the local system.
⚡ OnError	Occurs when an client operation fails.
⚡ OnRead	Occurs when data is available to be read from the client.
⚡ OnTimeout	Occurs when a blocking operation fails to complete before the timeout period elapses.
⚡ OnWrite	Occurs when data can be written to the client.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.OnCancel Event

Occurs when a blocking client operation is canceled.

[Visual Basic]

```
Public Event OnCancel As EventHandler
```

[C#]

```
public event EventHandler OnCancel;
```

Remarks

The **OnCancel** event is generated when a blocking client operation, such as sending or receiving data, is canceled with the **Cancel** method. To assist in determining which operation was canceled, check the value of the **Status** property.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.OnConnect Event

Occurs when a connection is established with the remote host.

[Visual Basic]

```
Public Event OnConnect As EventHandler
```

[C#]

```
public event EventHandler OnConnect;
```

Remarks

The **OnConnect** event occurs when a connection is made with a remote host as a result of a **Connect** method call. When the **Connect** method is called and the **Blocking** property is set to **false**, a socket is created but the connection is not actually established until after this event occurs. Between the time connection process is started and this event fires, no operation may be performed on the client other than calling the **Disconnect** method.

This event is only generated if the client is in non-blocking mode.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.OnDisconnect Event

Occurs when the remote host disconnects from the local system.

[Visual Basic]

```
Public Event OnDisconnect As EventHandler
```

[C#]

```
public event EventHandler OnDisconnect;
```

Remarks

The **OnDisconnect** event occurs when the remote host closes its connection, terminating the client session with the application. Because there may still be data in the client receive buffers, you should continue to read data from the client until the **Read** method returns a value of 0. Once all of the data has been read, you should call the **Disconnect** method to close the local socket and release the resources allocated for the client.

This event is only generated if the client is in non-blocking mode.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.OnError Event

Occurs when an client operation fails.

```
[Visual Basic]
Public Event OnError As OnErrorEventHandler
```

```
[C#]
public event OnErrorEventHandler OnError;
```

Event Data

The event handler receives an argument of type WhoisClient.ErrorEventArgs containing data related to this event. The following **WhoisClient.ErrorEventArgs** properties provide information specific to this event.

Property	Description
Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Remarks

The **OnError** event occurs when a client operation fails.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.ErrorEventArgs Class

Provides data for the OnError event.

For a list of all members of this type, see WhoisClient.ErrorEventArgs Members.

System.Object
  System.EventArgs
    SocketTools.WhoisClient.ErrorEventArgs

[Visual Basic]

```
Public Class WhoisClient.ErrorEventArgs
    Inherits EventArgs
```

[C#]

```
public class WhoisClient.ErrorEventArgs : EventArgs
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

ErrorEventArgs specifies the numeric error code and a description of the error that has occurred.

An OnError event occurs when a method fails.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WhoisClient (in SocketTools.WhoisClient.dll)

See Also

WhoisClient.ErrorEventArgs Members | SocketTools Namespace

WhoisClient.ErrorEventArgs Members

WhoisClient.ErrorEventArgs overview

Public Instance Constructors

WhoisClient.ErrorEventArgs Constructor	Initializes a new instance of the WhoisClient.ErrorEventArgs class.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

WhoisClient.ErrorEventArgs Class | SocketTools Namespace

WhoisClient.ErrorEventArgs Constructor

Initializes a new instance of the WhoisClient.ErrorEventArgs class.

[Visual Basic]

```
Public Sub New()
```

[C#]

```
public WhoisClient.ErrorEventArgs();
```

See Also

WhoisClient.ErrorEventArgs Class | SocketTools Namespace

WhoisClient.ErrorEventArgs Properties

The properties of the **WhoisClient.ErrorEventArgs** class are listed below. For a complete list of **WhoisClient.ErrorEventArgs** class members, see the WhoisClient.ErrorEventArgs Members topic.

Public Instance Properties

Description	Gets a value which describes the last error that has occurred.
Error	Gets a value which specifies the last error that has occurred.

See Also

WhoisClient.ErrorEventArgs Class | SocketTools Namespace

WhoisClient.ErrorEventArgs.Description Property

Gets a value which describes the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Description As String
```

[C#]

```
public string Description {get;}
```

Property Value

A string which describes the last error that has occurred.

See Also

WhoisClient.ErrorEventArgs Class | SocketTools Namespace | Error Property

WhoisClient.ErrorEventArgs.Error Property

Gets a value which specifies the last error that has occurred.

[Visual Basic]

```
Public ReadOnly Property Error As ErrorCode
```

[C#]

```
public WhoisClient.ErrorCode Error {get;}
```

Property Value

ErrorCode enumeration which specifies the error.

See Also

WhoisClient.ErrorEventArgs Class | SocketTools Namespace | Description Property

WhoisClient.OnRead Event

Occurs when data is available to be read from the client.

[Visual Basic]
```
Public Event OnRead As EventHandler
```

[C#]
```
public event EventHandler OnRead;
```

Remarks

The **OnRead** event occurs when data is available to be read from the client. This event is level-triggered, which means that once this event fires, it will not occur again until some data has been read from the client. This design prevents an application from being flooded with event notifications. It is recommended that your application read all of the available data from the server and store it in a local buffer for processing. See the example below.

This event is only generated if the client is in non-blocking mode.

Example

```
Private Sub Socket_OnRead(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Socket.OnRead
    Dim strBuffer As String
    Dim nRead As Integer

    Do
        ' Read up to m_nBufferSize bytes of data from the server
        nRead = Socket.Read(strBuffer, m_nBufferSize)

        If nRead > 0 Then
            ' Append the data to an internal buffer for processing
            m_dataBuffer = m_dataBuffer + strBuffer
        End If
    Loop Until nRead < 1

    ProcessData()
End Sub
```

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.OnTimeout Event

Occurs when a blocking operation fails to complete before the timeout period elapses.

[Visual Basic]

```
Public Event OnTimeout As EventHandler
```

[C#]

```
public event EventHandler OnTimeout;
```

Remarks

The **OnTimeout** event occurs when a blocking operation, such as sending or receiving data on the client, fails to complete before the specified timeout period elapses. The timeout period for a blocking operation can be adjusted by setting the **Timeout** property.

This event is only generated if the client is in blocking mode.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.OnWrite Event

Occurs when data can be written to the client.

[Visual Basic]
```
Public Event OnWrite As EventHandler
```

[C#]
```
public event EventHandler OnWrite;
```

Remarks

The **OnWrite** event occurs when the application can write data to the client. This event will typically occur when a connection is first established with the remote host, and after the **Write** method has failed because there was insufficient memory available in the client send buffers. In the second case, when some of the buffered data has been successfully sent to the remote host and there is space available in the send buffers, this event is used to signal the application that it may attempt to send more data.

This event is only generated if the client is in non-blocking mode.

See Also

WhoisClient Class | SocketTools Namespace

WhoisClient.ErrorCode Enumeration

Specifies the error codes returned by the WhoisClient class.

[Visual Basic]
```
Public Enum WhoisClient.ErrorCode
```

[C#]
```
public enum WhoisClient.ErrorCode
```

Remarks

The WhoisClient class uses the **ErrorCode** enumeration to specify what error has occurred when a method fails. The current error code may be determined by checking the value of the **LastError** property.

Note that the last error code is only meaningful if the previous operation has failed.

Members

Member Name	Description
errorNone	No error.
errorNotHandleOwner	Handle not owned by the current thread.
errorFileNotFound	The specified file or directory does not exist.
errorFileNotCreated	The specified file could not be created.
errorOperationCanceled	The blocking operation has been canceled.
errorInvalidFileType	The specified file is a block or character device, not a regular file.
errorInvalidDevice	The specified file type is invalid or not a regular file.
errorTooManyParameters	The maximum number of function parameters has been exceeded.
errorInvalidFileName	The specified file name contains invalid characters or is too long.
errorInvalidFileHandle	Invalid file handle passed to function.
errorFileReadFailed	Unable to read data from the specified file.
errorFileWriteFailed	Unable to write data to the specified file.
errorOutOfMemory	Out of memory.
errorAccessDenied	Access denied.
errorInvalidParameter	Invalid argument passed to function.
errorClipboardUnavailable	The system clipboard is currently unavailable.
errorClipboardEmpty	The system clipboard is empty or does not contain any text data.
errorFileEmpty	The specified file does not contain any data.
errorFileExists	The specified file already exists.

errorEndOfFile	End of file.
errorDeviceNotFound	The specified device could not be found.
errorDirectoryNotFound	The specified directory could not be found.
errorInvalidbuffer	Invalid memory address passed to function.
errorBufferTooSmall	The specified buffer is not large enough to contain the data.
errorNoHandles	No more handles are available to this process.
errorOperationWouldBlock	The specified operation would block the current thread.
errorOperationInProgress	A blocking operation is currently in progress.
errorAlreadyInProgress	The specified operation is already in progress.
errorInvalidHandle	Invalid handle passed to function.
errorInvalidAddress	Invalid network address specified.
errorInvalidSize	Datagram is too large to fit in specified buffer.
errorInvalidProtocol	Invalid network protocol specified.
errorProtocolNotAvailable	The specified network protocol is not available.
errorProtocolNotSupported	The specified protocol is not supported.
errorSocketNotSupported	The specified socket type is not supported.
errorInvalidOption	The specified option is invalid.
errorProtocolFamily	Specified protocol family is not supported.
errorProtocolAddress	The specified address is invalid for this protocol family.
errorAddressInUse	The specified address is in use by another process.
errorAddressUnavailable	The specified address cannot be assigned.
errorNetworkUnavailable	The networking subsystem is unavailable.
errorNetworkUnreachable	The specified network is unreachable.
errorNetworkReset	Network dropped connection on remote reset.
errorConnectionAborted	Connection was aborted due to timeout or other failure.
errorConnectionReset	Connection was reset by remote network.
errorOutOfBuffers	No buffer space is available.
errorAlreadyConnected	Connection already established with remote host.
errorNotConnected	No connection established with remote host.
errorConnectionShutdown	Unable to send or receive data after connection shutdown.
errorOperationTimeout	The specified operation has timed out.
errorConnectionRefused	The connection has been refused by the remote host.

errorHostUnavailable	The specified host is unavailable.
errorHostUnreachable	Remote host is unreachable.
errorTooManyProcesses	Too many processes are using the networking subsystem.
errorTooManyThreads	Too many threads have been created by the current process.
errorTooManySessions	Too many client sessions have been created by the current process.
errorInternalFailure	An unexpected internal error has occurred.
errorNetworkNotReady	Network subsystem is not ready for communication.
errorInvalidVersion	This version of the operating system is not supported.
errorNetworkNotInitialized	The networking subsystem has not been initialized.
errorRemoteShutdown	The remote host has initiated a graceful shutdown sequence.
errorInvalidHostName	The specified hostname is invalid or could not be resolved.
errorHostNameNotFound	The specified hostname could not be found.
errorHostNameRefused	Unable to resolve hostname, request refused.
errorHostNameNotResolved	Unable to resolve hostname, no address for specified host.
errorInvalidLicense	The license for this product is invalid.
errorProductNotLicensed	This product is not licensed to perform this operation.
errorNotImplemented	This function has not been implemented on this platform.
errorUnknownLocalhost	Unable to determine local host name.
errorInvalidHostAddress	Invalid host address specified.
errorInvalidServicePort	Invalid service port number specified.
errorInvalidServiceName	Invalid or unknown service name specified.
errorInvalidEventId	Invalid event identifier specified.
errorOperationNotBlocking	No blocking operation in progress on this socket.
errorSecurityNotInitialized	Unable to initialize security interface for this process.
errorSecurityContext	Unable to establish security context for this session.
errorSecurityCredentials	Unable to open certificate store or establish security credentials.
errorSecurityCertificate	Unable to validate the certificate chain for this

	session.
errorSecurityDecryption	Unable to decrypt data stream.
errorSecurityEncryption	Unable to encrypt data stream.
errorOperationNotSupported	The specified operation is not supported.
errorInvalidProtocolVersion	Invalid application protocol version specified.
errorNoServerResponse	No data returned from server.
errorInvalidServerResponse	Invalid data returned from server.
errorUnexpectedServerResponse	Unexpected response code returned from server.
errorServerTransactionFailed	Server transaction failed.
errorServiceUnavailable	The service is currently unavailable.
errorServiceNotReady	The service is not ready, try again later.
errorServerResyncFailed	Unable to resynchronize with server.
errorInvalidProxyType	Invalid proxy server type specified.
errorProxyRequired	Resource must be accessed through specified proxy.
errorInvalidProxyLogin	Unable to login to proxy server using specified credentials.
errorProxyResyncFailed	Unable to resynchronize with proxy server.
errorInvalidCommand	Invalid command specified.
errorInvalidCommandParameter	Invalid command parameter specified.
errorInvalidCommandSequence	Invalid command sequence specified.
errorCommandNotImplemented	Specified command not implemented on this server.
errorCommandNotAuthorized	Specified command not authorized for the current user.
errorCommandAborted	Specified command was aborted by the remote host.
errorOptionNotSupported	The specified option is not supported on this server.
errorRequestNotCompleted	The current client request has not been completed.
errorInvalidUserName	The specified username is invalid.
errorInvalidPassword	The specified password is invalid.
errorInvalidAccount	The specified account name is invalid.
errorAccountRequired	Account name has not been specified.
errorInvalidAuthenticationType	Invalid authentication protocol specified.
errorAuthenticationRequired	User authentication is required.
errorProxyAuthenticationRequired	Proxy authentication required.

errorAlreadyAuthenticated	User has already been authenticated.
errorAuthenticationFailed	Unable to authenticate the specified user.
errorNetworkAdapter	Unable to determine network adapter configuration.
errorInvalidRecordType	Invalid record type specified.
errorInvalidRecordName	Invalid record name specified.
errorInvalidRecordData	Invalid record data specified.
errorConnectionOpen	Data connection already established.
errorConnectionClosed	Server closed data connection.
errorConnectionPassive	Data connection is passive.
errorConnectionFailed	Unable to open data connection to server.
errorInvalidSecurityLevel	Data connection cannot be opened with this security setting.
errorCachedTLSRequired	Data connection requires cached TLS session.
errorDataReadOnly	Data connection is read-only.
errorDataWriteOnly	Data connection is write-only.
errorEndOfData	End of data.
errorRemoteFileUnavailable	Remote file is unavailable.
errorInsufficientStorage	Insufficient storage on server.
errorStorageallocation	File exceeded storage allocation on server.
errorDirectoryExists	The specified directory already exists.
errorDirectoryEmpty	No files returned by the server for the specified directory.
errorEndOfDirectory	End of directory listing.
errorUnknownDirectoryFormat	Unknown directory format.
errorInvalidResource	Invalid resource name specified.
errorResourceRedirected	The specified resource has been redirected.
errorResourceRestricted	Access to this resource has been restricted.
errorResourceNotModified	The specified resource has not been modified.
errorResourceNotFound	The specified resource cannot be found.
errorResourceConflict	Request could not be completed due to the current state of the resource.
errorResourceRemoved	The specified resource has been permanently removed from this server.
errorContentLengthRequired	Request must include the content length.
errorRequestPrecondition	Request could not be completed due to server precondition.
errorUnsupportedMediaType	Request specified an unsupported media type.

errorInvalidContentRange	Content range specified for this resource is invalid.
errorInvalidMessagePart	Message is not multipart or an invalid message part was specified.
errorInvalidMessageHeader	The specified message header is invalid or has not been defined.
errorInvalidMessageBoundary	The multipart message boundary has not been defined.
errorNoFileAttachment	The current message part does not contain a file attachment.
errorUnknownFileType	The specified file type could not be determined.
errorDataNotEncoded	The specified data block could not be encoded.
errorDataNotDecoded	The specified data block could not be decoded.
errorFileNotEncoded	The specified file could not be encoded.
errorFileNotDecoded	The specified file could not be decoded.
errorNoMessageText	No message text.
errorInvalidCharacterSet	Invalid character set specified.
errorInvalidEncodingType	Invalid encoding type specified.
errorInvalidMessageNumber	Invalid message number specified.
errorNoReturnAddress	No valid return address specified.
errorNoValidRecipients	No valid recipients specified.
errorInvalidRecipient	The specified recipient address is invalid.
errorRelayNotAuthorized	The specified domain is invalid or server will not relay messages.
errorMailboxUnavailable	Specified mailbox is currently unavailable.
errorMailboxReadOnly	The selected mailbox cannot be modified.
errorMailboxNotSelected	No mailbox has been selected.
errorInvalidMailbox	Specified mailbox is invalid.
errorInvalidDomain	The specified domain name is invalid or not recognized.
errorInvalidSender	The specified sender address is invalid or not recognized.
errorMessageNotDelivered	Message not delivered to any of the specified recipients.
errorEndOfMessageData	No more message data available to be read
errorInvalidmessageSize	The specified message size is invalid.
errorMessageNotCreated	The message could not be created in the specified mailbox.
errorNoMoreMailboxes	No more mailboxes exist on this server.

errorInvalidEmulationType	The specified terminal emulation type is invalid.
errorInvalidFontHandle	The specified font handle is invalid.
errorInvalidFontName	The specified font name is invalid or unavailable.
errorInvalidPacketSize	The specified packet size is invalid.
errorInvalidPacketData	The specified packet data is invalid.
errorInvalidPacketId	The unique packet identifier is invalid.
errorPacketTTLExpired	The specified packet time-to-live period has expired.
errorInvalidNewsGroup	Invalid newsgroup specified.
errorNoNewsgroupSelected	No newsgroup selected.
errorEmptyNewsgroup	No articles in specified newsgroup.
errorInvalidArticle	Invalid article number specified.
errorNoArticleSelected	No article selected in the current newsgroup.
errorFirstArticle	First article in current newsgroup.
errorLastArticle	Last article in current newsgroup.
errorArticleExists	Unable to transfer article, article already exists.
errorArticleRejected	Unable to transfer article, article rejected.
errorArticleTransferFailed	Article transfer failed.
errorArticlePostingDenied	Posting is not permitted on this server.
errorArticlePostingFailed	Unable to post article on this server.
errorInvalidDateFormat	The specified date format is not recognized.
errorFeatureNotSupported	The specified feature is not supported on this server.
errorInvalidFormHandle	The specified form handle is invalid or a form has not been created.
errorInvalidFormAction	The specified form action is invalid or has not been specified.
errorInvalidFormMethod	The specified form method is invalid or not supported.
errorInvalidFormType	The specified form type is invalid or not supported.
errorInvalidFormField	The specified form field name is invalid or does not exist.
errorEmptyForm	The specified form does not contain any field values.
errorMaximumConnections	The maximum number of client connections exceeded.
errorThreadCreationFailed	Unable to create a new thread for the current process.
errorInvalidThreadHandle	The specified thread handle is no longer valid.

errorThreadTerminated	The specified thread has been terminated.
errorThreadDeadlock	The operation would result in the current thread becoming deadlocked.
errorInvalidClientMoniker	The specified moniker is not associated with any client session.
errorClientMonikerExists	The specified moniker has been assigned to another client session.
errorServerInactive	The specified server is not listening for client connections.
errorServerSuspended	The specified server is suspended and not accepting client connections.
errorNoMessageStore	No message store has been specified.
errorMessageStoreChanged	The message store has changed since it was last accessed.
errorMessageNotFound	No message was found that matches the specified criteria.
errorMessageDeleted	The specified message has been deleted.
errorFileChecksumMismatch	The local and remote file checksums do not match.
errorFileSizeMismatch	The local and remote file sizes do not match.
errorInvalidFeedUrl	The news feed URL is invalid or specifies an unsupported protocol.
errorInvalidFeedFormat	The internal format of the news feed is invalid.
errorInvalidFeedVersion	This version of the news feed is not supported.
errorChannelEmpty	There are no valid items found in this news feed.
errorInvalidItemNumber	The specified channel item identifier is invalid.
errorItemNotFound	The specified channel item could not be found.
errorItemEmpty	The specified channel item does not contain any data.
errorInvalidItemProperty	The specified item property name is invalid.
errorItemPropertyNotFound	The specified item property has not been defined.
errorInvalidChannelTitle	The channel title is invalid or has not been defined.
errorInvalidChannelLink	The channel hyperlink is invalid or has not been defined.
errorInvalidChannelDescription	The channel description is invalid or has not been defined.
errorInvalidItemText	The description for an item is invalid or has not been defined.
errorInvalidItemLink	The hyperlink for an item is invalid or has not been defined.

errorInvalidServiceType	The specified service type is invalid.
errorServiceSuspended	Access to the specified service has been suspended.
errorServiceRestricted	Access to the specified service has been restricted.
errorInvalidProviderName	The specified provider name is invalid or unknown.
errorInvalidPhoneNumber	The specified phone number is invalid or not supported in this region.
errorGatewayNotFound	A message gateway cannot be found for the specified provider.
errorMessageTooLong	The message exceeds the maximum number of characters permitted.
errorInvalidProviderData	The request returned invalid or incomplete service provider data.
errorInvalidGatewayData	The request returned invalid or incomplete message gateway data.
errorMultipleProviders	The request has returned multiple service providers.
errorProviderNotFound	The specified service provider could not be found.
errorInvalidMessageService	The specified message is not supported with this service type.
errorInvalidMessageFormat	The specified message format is invalid.
errorInvalidConfiguration	The specified configuration options are invalid.
errorServerActive	The requested action is not permitted while the server is active.
errorServerPortBound	Unable to obtain exclusive use of the specified local port.
errorInvalidClientSession	The specified client identifier is invalid for this session.
errorClientNotIdentified	The specified client has not provided user credentials.
errorInvalidClientState	The requested action cannot be performed at this time.
errorInvalidResultCode	The specified result code is not valid for this protocol
errorCommandRequired	The specified command is required and cannot be disabled.
errorCommandDisabled	The specified command has been disabled.
errorCommandSequence	The command cannot be processed at this time.
errorCommandCompleted	The previous command has completed.
errorInvalidProgramName	The specified program name is invalid or unrecognized.

errorInvalidRequestHeader	The request header contains one or more invalid values.
errorInvalidVirtualHost	The specified virtual host name is invalid.
errorVirtualHostNotFound	The specified virtual host does not exist.
errorTooManyVirtualHosts	Too many virtual hosts created for this server.
errorInvalidVirtualPath	The specified virtual path name is invalid.
errorVirtualPathNotFound	The specified virtual path does not exist.
errorTooManyVirtualPaths	Too many virtual paths created for this server.
errorInvalidTask	The asynchronous task identifier is invalid.
errorTaskActive	The asynchronous task has not finished.
errorTaskQueued	The asynchronous task has been queued.
errorTaskSuspended	The asynchronous task has been suspended.
errorTaskFinished	The asynchronous task has finished.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WhoisClient (in SocketTools.WhoisClient.dll)

See Also

SocketTools Namespace | LastError Property | LastErrorString Property | OnError Event

WhoisClient.TraceOptions Enumeration

Specifies the logging options that the WhoisClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]
```
<Flags>
Public Enum WhoisClient.TraceOptions
```

[C#]
```
[Flags]
public enum WhoisClient.TraceOptions
```

Remarks

The WhoisClient class uses the **TraceOptions** enumeration to specify what kind of debugging information is written to the trace logfile. These options are only meaningful when trace logging is enabled by setting the **Trace** property to **true**.

Members

Member Name	Description	Value
traceDefault	The default trace logging option. This is the same as specifying the **traceInfo** option.	0
traceInfo	All network function calls are written to the trace file. This is the default value.	0
traceError	Only those network function calls which fail are recorded in the trace file.	1
traceWarning	Only those network function calls which fail, or return values which indicate a warning, are recorded in the trace file.	2
traceHexDump	All network function calls are written to the trace file, plus all the data that is sent or received is displayed, in both ASCII and hexadecimal format.	4
traceProcess	All function calls in the current process are logged, rather than only those functions in the current thread. This option is useful for multithreaded applications that are using worker threads.	4096

Requirements

Namespace: SocketTools

Assembly: SocketTools.WhoisClient (in SocketTools.WhoisClient.dll)

See Also

SocketTools Namespace

WhoisClient.WhoisOptions Enumeration

Specifies the options that the WhoisClient class supports.

This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

[Visual Basic]

```
<Flags>
Public Enum WhoisClient.WhoisOptions
```

[C#]

```
[Flags]
public enum WhoisClient.WhoisOptions
```

Remarks

The WhoisClient class uses the **WhoisOptions** enumeration to specify one or more options to be used when establishing a connection with a remote host. Multiple options may be specified if necessary.

There are currently no additional options for WhoisClient class. This enumeration is provided for future expansion.

Members

Member Name	Description	Value
optionNone	No option specified.	0
optionDefault	The default connection option. This is the same as specifying **optionNone**.	0
optionFreeThread	This option specifies that class methods may be called from any thread, and not only the thread that established the connection. Using this option disables certain internal safety checks that are made by the class and may result in unexpected behavior unless you ensure that access to the class instance is synchronized across multiple threads.	524288

Requirements

Namespace: SocketTools

Assembly: SocketTools.WhoisClient (in SocketTools.WhoisClient.dll)

See Also

SocketTools Namespace

WhoisClient.WhoisSearchType Enumeration

Specifies the search options supported by the WhoisClient class.

[Visual Basic]

```
Public Enum WhoisClient.WhoisSearchType
```

[C#]

```
public enum WhoisClient.WhoisSearchType
```

Remarks

The **WhoisSearchType** enumeration specifies the types of queries that can be performed using the **Search** method. If you wish to perform a more complex query using the syntax outlined in RFC 954, specify a search type of **searchAny** and then provide the search string value that you want to submit.

Members

Member Name	Description
searchAny	Search for any value that matches the given keyword value. This is the default search type.
searchHandle	Search for a handle that matches the given keyword value.
searchName	Search for a user name that matches the given keyword value.
searchMailbox	Search for a user mailbox that matches the given keyword value.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WhoisClient (in SocketTools.WhoisClient.dll)

See Also

SocketTools Namespace

WhoisClient.WhoisStatus Enumeration

Specifies the status values that may be returned by the WhoisClient class.

[Visual Basic]

```
Public Enum WhoisClient.WhoisStatus
```

[C#]

```
public enum WhoisClient.WhoisStatus
```

Remarks

The WhoisClient class uses the **WhoisStatus** enumeration to identify the current status of the client.

Members

Member Name	Description
statusUnused	A client session has not been created. Attempts to perform any network operations, such as sending or receiving data, will generate an error.
statusIdle	A client session has been created, but is not currently in use. A blocking socket operation can be executed at this point.
statusConnect	The client is in the process of establishing a connection with a remote host.
statusRead	The client is in the process of receiving data from a remote host.
statusWrite	The client is in the process of sending data to a remote host.
statusDisconnect	The client session is being closed and subsequent attempts to access the client will result in an error.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WhoisClient (in SocketTools.WhoisClient.dll)

See Also

SocketTools Namespace

WhoisClient.OnErrorEventHandler Delegate

Represents the method that will handle the OnError event.

[Visual Basic]

```
Public Delegate Sub WhoisClient.OnErrorEventHandler( _
   ByVal sender As Object, _
   ByVal e As ErrorEventArgs _
)
```

[C#]

```
public delegate void WhoisClient.OnErrorEventHandler(
      object sender,
      ErrorEventArgs e
   );
```

Parameters

sender

The source of the event.

e

An ErrorEventArgs that contains the event data.

Remarks

When you create an **OnErrorEventHandler** delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, until you remove the delegate.

Note that the declaration of your event handler must have the same parameters as the **OnErrorEventHandler** delegate declaration.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WhoisClient (in SocketTools.WhoisClient.dll)

See Also

SocketTools Namespace

WhoisClient.RuntimeLicenseAttribute Class

Attribute that defines the runtime license key for the class.

For a list of all members of this type, see WhoisClient.RuntimeLicenseAttribute Members.

System.Object
  System.Attribute
    SocketTools.WhoisClient.RuntimeLicenseAttribute

[Visual Basic]

```
<AttributeUsage(ValidOn:=AttributeTargets.Assembly, AllowMultiple:=False,
   Inherited:=True)>
Public Class WhoisClient.RuntimeLicenseAttribute
    Inherits Attribute
```

[C#]

```
[AttributeUsage(ValidOn=AttributeTargets.Assembly, AllowMultiple=False,
   Inherited=True)]
public class WhoisClient.RuntimeLicenseAttribute : Attribute
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

The RuntimeLicense attribute is used to define the runtime license key that will be used when an instance of the class is created. This attribute is defined in the assembly information module for the language, such as AssemblyInfo.cs when programming C#. The runtime license key must be defined if you wish to redistribute your application.

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the WhoisClient class.

Example

```
<Assembly: SocketTools.WhoisClient.RuntimeLicense("abcdefghijklmnop")>
```

```
[assembly: SocketTools.WhoisClient.RuntimeLicense("abcdefghijklmnop")]
```

Requirements

Namespace: SocketTools

Assembly: SocketTools.WhoisClient (in SocketTools.WhoisClient.dll)

See Also

WhoisClient.RuntimeLicenseAttribute Members | SocketTools Namespace

WhoisClient.RuntimeLicenseAttribute Members

WhoisClient.RuntimeLicenseAttribute overview

Public Instance Constructors

WhoisClient.RuntimeLicenseAttribute Constructor	Constructor for the RuntimeLicense attribute which defines the runtime license key.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

Public Instance Methods

Equals (inherited from Attribute)	
GetHashCode (inherited from Attribute)	Returns the hash code for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IsDefaultAttribute (inherited from Attribute)	When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.
Match (inherited from Attribute)	When overridden in a derived class, returns a value indicating whether this instance equals a specified object.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

WhoisClient.RuntimeLicenseAttribute Class | SocketTools Namespace

WhoisClient.RuntimeLicenseAttribute Constructor

Constructor for the RuntimeLicense attribute which defines the runtime license key.

[Visual Basic]

```
Public Sub New( _
   ByVal licenseKey As String _
)
```

[C#]

```
public WhoisClient.RuntimeLicenseAttribute(
   string licenseKey
);
```

Parameters

licenseKey

A string argument which specifies the runtime license key which will be used to initialize the class library.

Remarks

The runtime license key for your copy of SocketTools can be generated using the License Manager utility that is included with the product. Note that if you have installed an evaluation license, you will not have a runtime license key and cannot redistribute any applications which use the WhoisClient class.

See Also

WhoisClient.RuntimeLicenseAttribute Class | SocketTools Namespace

WhoisClient.RuntimeLicenseAttribute Properties

The properties of the **WhoisClient.RuntimeLicenseAttribute** class are listed below. For a complete list of **WhoisClient.RuntimeLicenseAttribute** class members, see the WhoisClient.RuntimeLicenseAttribute Members topic.

Public Instance Properties

LicenseKey	Returns the value of the runtime license key.
TypeId (inherited from Attribute)	When implemented in a derived class, gets a unique identifier for this Attribute.

See Also

WhoisClient.RuntimeLicenseAttribute Class | SocketTools Namespace

WhoisClient.RuntimeLicenseAttribute.LicenseKey Property

Returns the value of the runtime license key.

[Visual Basic]

```
Public Property LicenseKey As String
```

[C#]

```
public string LicenseKey {get; set;}
```

Property Value

A string which contains the runtime license key.

See Also

WhoisClient.RuntimeLicenseAttribute Class | SocketTools Namespace

WhoisClientException Class

The exception that is thrown when a client error occurs.

For a list of all members of this type, see WhoisClientException Members.

System.Object
  System.Exception
    System.ApplicationException
      SocketTools.WhoisClientException

[Visual Basic]

```
Public Class WhoisClientException
    Inherits ApplicationException
```

[C#]

```
public class WhoisClientException : ApplicationException
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

A WhoisClientException is thrown by the WhoisClient class when an error occurs.

The default constructor for the WhoisClientException class sets the **ErrorCode** property to the last client error that occurred.

Requirements

Namespace: SocketTools

Assembly: SocketTools.WhoisClient (in SocketTools.WhoisClient.dll)

See Also

WhoisClientException Members | SocketTools Namespace

WhoisClientException Members

WhoisClientException overview

Public Instance Constructors

WhoisClientException	Overloaded. Initializes a new instance of the WhoisClientException class.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

WhoisClientException Class | SocketTools Namespace

WhoisClientException Constructor

Initializes a new instance of the WhoisClientException class with the last network error code.

Overload List

Initializes a new instance of the WhoisClientException class with the last network error code.

public WhoisClientException();

Initializes a new instance of the WhoisClientException class with a specified error number.

public WhoisClientException(int);

Initializes a new instance of the WhoisClientException class with a specified error message.

public WhoisClientException(string);

Initializes a new instance of the WhoisClientException class with a specified error message and a reference to the inner exception that is the cause of this exception.

public WhoisClientException(string,Exception);

See Also

WhoisClientException Class | SocketTools Namespace

WhoisClientException Constructor ()

Initializes a new instance of the WhoisClientException class with the last network error code.

[Visual Basic]

```
Overloads Public Sub New()
```

[C#]

```
public WhoisClientException();
```

Remarks

The ctor constructor sets the **ErrorCode** property to the last client error that occurred. For more information about the errors that may occur, refer to the WhoisClient.ErrorCode enumeration.

See Also

WhoisClientException Class | SocketTools Namespace | WhoisClientException Constructor Overload List

WhoisClientException Constructor (String)

Initializes a new instance of the WhoisClientException class with a specified error message.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String _
)
```

[C#]

```
public WhoisClientException(
   string message
);
```

Parameters

message

The error message that explains the reason for the exception.

Remarks

The content of the ***message*** parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

See Also

WhoisClientException Class | SocketTools Namespace | WhoisClientException Constructor Overload List

WhoisClientException Constructor (String, Exception)

Initializes a new instance of the WhoisClientException class with a specified error message and a reference to the inner exception that is the cause of this exception.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal message As String, _
   ByVal innerException As Exception _
)
```

[C#]

```
public WhoisClientException(
   string message,
   Exception innerException
);
```

Parameters

message

The error message that explains the reason for the exception.

innerException

The exception that is the cause of the current exception. If the ***innerException*** parameter is not a null reference, the current exception is raised in a **catch** block that handles the inner exception.

Remarks

The content of the message parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the **InnerException** property. The **InnerException** property returns the same value that is passed into the constructor, or a null reference if the **InnerException** property does not supply the inner exception value to the constructor.

See Also

WhoisClientException Class | SocketTools Namespace | WhoisClientException Constructor Overload List

WhoisClientException Constructor (Int32)

Initializes a new instance of the WhoisClientException class with a specified error number.

[Visual Basic]

```
Overloads Public Sub New( _
   ByVal code As Integer _
)
```

[C#]

```
public WhoisClientException(
   int code
);
```

Parameters

code

An integer value which specifies an error code.

Remarks

This constructor sets the **ErrorCode** property to the specified error code. For more information about the errors that may occur, refer to the WhoisClient.ErrorCode enumeration.

See Also

WhoisClientException Class | SocketTools Namespace | WhoisClientException Constructor Overload List

WhoisClientException Properties

The properties of the **WhoisClientException** class are listed below. For a complete list of **WhoisClientException** class members, see the WhoisClientException Members topic.

Public Instance Properties

ErrorCode	Gets a value which specifies the error that caused the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message	Gets a value which describes the error that caused the exception.
Number	Gets a value which specifies the numeric value of the error that caused the exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

See Also

WhoisClientException Class | SocketTools Namespace

WhoisClientException.ErrorCode Property

Gets a value which specifies the error that caused the exception.

[Visual Basic]
```
Public ReadOnly Property ErrorCode As ErrorCode
```

[C#]
```
public WhoisClient.ErrorCode ErrorCode {get;}
```

Property Value

Returns a WhoisClient.ErrorCode enumeration value which specifies the error code.

Remarks

The **ErrorCode** property returns the error code that specifies the cause of the exception.

The default constructor for the WhoisClientException class sets the error code to the last network error that occurred. For more information about the errors that may occur, refer to the WhoisClient.ErrorCode enumeration.

See Also

WhoisClientException Class | SocketTools Namespace

WhoisClientException.Message Property

Gets a value which describes the error that caused the exception.

[Visual Basic]

```
Overrides Public ReadOnly Property Message As String
```

[C#]

```
public override string Message {get;}
```

Property Value

A string which describes the error that caused the exception.

Remarks

The **Message** property returns a string which describes the error that caused the exception.

See Also

WhoisClientException Class | SocketTools Namespace

WhoisClientException.Number Property

Gets a value which specifies the numeric value of the error that caused the exception.

[Visual Basic]

```
Public ReadOnly Property Number As Integer
```

[C#]

```
public int Number {get;}
```

Property Value

An integer value that specifies the error that caused the exception.

Remarks

The **Number** property returns an integer value which specifies the numeric value of the error that caused the exception. This value is the same as the values returned by the Windows Sockets API. For more information about socket error codes, see the Windows Socket Version 2 API error code documentation in MSDN.

See Also

WhoisClientException Class | SocketTools Namespace

WhoisClientException Methods

The methods of the **WhoisClientException** class are listed below. For a complete list of **WhoisClientException** class members, see the WhoisClientException Members topic.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Creates and returns a string representation of the current exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

WhoisClientException Class | SocketTools Namespace

WhoisClientException.ToString Method

Creates and returns a string representation of the current exception.

[Visual Basic]

```
Overrides Public Function ToString() As String
```

[C#]

```
public override string ToString();
```

Return Value

A string representation of the current exception.

Remarks

The **ToString** method returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by **ToString** is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

This implementation of **ToString** obtains the numeric error code value and a description of the error that caused the current exception. If there is no error message or it is an empty string, then no error message is returned.

This method overrides **ApplicationException.ToString**.

See Also

WhoisClientException Class | SocketTools Namespace